НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

зі спеціальності 122 Комп’ютерні науки та інформаційні технології
за спеціалізацією Геометричне моделювання в інформаційних системах
на тему____Бібліотека методів C++ для роботи з сервером електронної пошти_____
за протоколом IMAP_____
_____

Виконав (-ла): студент (-ка) ___4__курсу, групи_ТР-62___

_____Пляс Ольга Олександрівна_____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник ____доцент Кузьменко І.М._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

зі спеціальності 122 Комп’ютерні науки та інформаційні технології
за спеціалізацією Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

_____
(прізвище, ім’я, по батькові)

1. Тема роботи ____Бібліотека методів C++ для роботи з сервером електронної пошти за протоколом IMAP_____
_____
_____
керівник роботи____доцент Кузьменко Ігор Миколайович_____
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи ____Стандартизовані відповіді сервера на запити__ клієнта протоколу IMAP4._____

_____

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)____Ознайомлення зі стандартними командами та відповідями поштового протоколу IMAP4, графічний опис та архітектура роботи програми на основі поштового протоколу IMAP4, демонстрація роботи розробленої бібліотеки методів на основі поштового протоколу IMAP4._____
_____
_____
_____
_____
_____

5. Перелік ілюстративного матеріалу
Діаграми для графічного опису системи : діагарама станів та діаграма компонентів. Приклади роботи консолі для демонстрації роботи методів бібліотеки у вигляді скріншотів рядків коду і їх результат у консолі._____
_____

6. Дата видачі завдання ”___”_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури системи		
4.	Розробка структур окремих підсистем		
5.	Програмна реалізація системи		
6.	Оформлення пояснювальної записки		
7.	Захист програмного продукту		
8.	Передзахист		
9.	Захист		

Студент _____ _____Пляс О.О._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Кузьменко І.М._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

В пропонованій дипломній роботі розроблено програмну бібліотеку методів для реалізації протоколу Internet Message Access Protocol 4rev1 мовою C++.

Метою роботи є розробка бібліотеки, яка, при її підключенні до програми чи іншого програмного продукту буде коректно виконувати роботу поштового протоколу Internet Message Access Protocol 4rev1.

В результаті було розроблено бібліотеку методів на основі стандартних команд протоколу Internet Message Access Protocol 4rev1, які дають максимальні можливості для роботи даного протоколу.

Пояснювальна записка містить 74 сторінок, 30 рисунків, 3 додатки.

Розроблена система призначення для підключення її до інших систем та програм, як модуля для реалізації поштового протоколу

ABSTRACT

In the offered diploma work the software library of methods developed in C ++ language for realization of the Internet Message Access Protocol 4rev1 protocol.

The purpose of this work is to develop a library that, when connected to a program or other software product, will correctly perform the work of the Internet Protocol Access Protocol 4rev1.

As a result, a library of methods was developed based on the standard commands of the Internet Message Access Protocol 4rev1, which provide maximum opportunities for the operation of this protocol.

The explanatory note contains 74 pages, 30 figures, 3 appendices.

A destination system has been developed to connect it to other systems and programs as a module for implementing the Internet Message Access Protocol 4rev1.

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

IMAP4 — Internet Message Access Protocol 4rev1.

POP3 — Post Office Protocol.

UTF7 - 7-bit Unicode Transformation Format

SSL – Security Socket Layer

TCP/IP – Transmission Contro; Protocol / Internet Protocol

SMTP – Simple Mail Transfer Protocol

ВСТУП

Наразі електронна пошта та листування міцно закріпилися в нашому житті.

Ми настільки звикли до того, що всі робочі, особисті та ділові питання ми можемо вирішити просто написавши листа на своєму комп’ютері, смартфоні чи планшеті. У кожного з нас є електронна адреса – без неї зараз ти є неповноцінною особою сучасного суспільства.

Чи могли про це подумати тоді, в 1965 році, задовго до появи інтернету та знайомих нам операційних систем, співробітники Массачусетського технологічного інститут - Ноель Моріс і Том Ван Влек, які написали програму MAIL для операційної системи CTSS (Compatible Time-Sharing System), встановлену на комп'ютері IBM 7090/7094.

Після цього електронна пошта пройшла чотири кроки розвитку. Спочатку користувачі могли користуватися програмою mail або її еквівалентами лише в межах одного мейнфрейма, простіше говорячи одного великого комп’ютера. Після цього з’явилася можливість надіслати повідомлення користувача на інший комп’ютер, для цього використовували щось схоже на сучасний імейл. Для цього використовувалися ім'я комп’ютеру та ім'я користувача на комп’ютері. Третім кроком стала можливість надсилати листи через третій комп’ютер, але для цього користувачеві потрібно було знати точну «дорогу», яку потрібно пройти листу до свого одержувача. І ось останній крок до становлення електронної пошти такою як ми її знаємо зараз стала поява DNS, і для назви комп’ютера, який вказували в імейл стали використовувати виділені сервери, на який не мали доступу користувачі, а в цей час вони працювали на своїх машинах. Тоді і з’явилися поштові протоколи, для роботи користувачів з цими серверами, серед яких і IMAP4, про який я й розкажу вам сьогодні.

Цей протокол є одним із найсучасніших на даний момент. Він дозволяє робити всі, вже знайомі нам, дії з електронною поштою, а саме: переглядання,

сортування, видалення, пошук по ключовому слову, підписання, відписання та інше.

У цьому документі ми зможемо побачити бібліотеку C++, яка повністю реалізує протокол та функції протоколу IMAP4, та зможе підключити його до будь-якої вашої програми. Ми розглянемо усі тонкості та особливості роботи протоколу IMAP4, які були необхідні для його програмної реалізації та подивимося приклад роботи цього протоколу.

1. Задача коректної роботи протоколу IMAP4

Метою роботи є розробка бібліотеки C++ для реалізації протоколу IMAP4, для подальшого підключення цього протоколу до програм.

Протокол IMAP4 має реалізовувати такі функції:

1) Встановлення сеансу з сервером;
2) Автентифікацію користувача;
3) Роботу з поштовим ящиком;
4) Роботу з папками (створення, видалення, перейменування, перегляд вмісту, отримання інформації про неї, вибір папки для подальшої роботи з нею);
5) Робота з повідомленнями (видалення, копіювання, пошук повідомлень, інформація про повідомлення, зберігання змін внесених в повідомлення);
6) Завершення сеансу з сервером;

Протокол IMAP4 працює за принципом запит-відповідь. Клієнт, тобто ми, як користувач надсилаємо запит на якусь дію з нашою поштовою скринькою, а сервер дає нам відповідь – надсилає дані по нашому запиту та результат завершення запиту. Наша бібліотека повинна вміти правильно розуміти та розшифровувати відповіді сервера на наші запити.

1.1. Вхідні дані

В першу чергу нашими вхідними даними є дані для встановлення сеансу з сервером, а саме хост та порт для під’єднання до сервера.

Для кожної команди протоколу ми маємо стандартизований запит, якого потрібно строго дотримуватися, інакше сервер не дасть відповідь.

Команда починається з ідентифікатора (частіше всього короткий набір символів, наприклад A0001, A0002), їх ще називають тегами. Вони створені для того, щоб на тег з конкретною командою приходила відповідь на саме цю

команду з таким самим тегом. Після тегу пишуть саму команду та відповідний йому аргумент. У кожної команди цей аргумент різний, у деяких його взагалі немає.

Спочатку варто розглянути всі команди які існують в протоколі IMAP4 і які функції вони виконують, адже саме на них базується вся робота з клієнтом нашого протоколу.

1) Команда Capability - запитує у сервера його список можливостей;

2) Команда Noop – робить періодичні пули, для підтримання зв’язку з сервером, щоб сеанс не розірвався по таймеру інтервала очікування, ще говорять що вона робить «нічого»;

3) Команда Logout – повідомляє серверу, що клієнт розриває з’єднання з сервером;

4) Команда StartTLS – вмикає шифрування з’єднання із сервером явно;

5) Команда Authenticate – надсилає серверу назву механізму автентифікації, яким клієнт хоче користуватися;

6) Команда Login – ідентифікує користувача, надсилає ім’я та пароль користувача для входу в поштову скриньку, також за замовчуванням надсилається команда Capability;

7) Команда Select – обирає поштову папку з поштової скриньки, щоб далі працювати з нею та мати доступ до її повідомлень;

8) Команда Examine – ця команда ідентична до команди Select, лише з єдиною відмінністю в тому, що папка яку ми обираємо ідентифікується, як папка лише для читання;

9) Команда Create – створює нову поштову папку та дає їй назву в нашій поштовій скоринці;

10) Команда Delete – видаляє обрану поштову папку та всі листи у ній, точніше спочатку переносить її в delete inbox і за позитивної відповіді сервера видаляє;

11) Команда Rename – змінює ім’я поштової папки;

12) Команда Subscribe – додає вказану поштову папку в список активних папок клієнта;

13) Команда Unsubscribe – видаляє вказану поштову папку зі списку активних папок клієнта;

14) Команда List – надсилає запит на отримання списку всіх поштових папок клієнта;

15) Команда Lsub – практично ідентична до команди List, окрім того що ми надсилаємо запит на отримання списку всіх поштових папок клієнта, активізованих командою Subscribe

16) Команда Status – надсилає запит про поточний стан обраної поштової скриньки;

17) Команда Append – додає аргумент літерал, як новий лист у кінець вказаної поштової папки;

18) Команда Check – встановлює контрольну точку, ще її називають чекпоінт, у вказаній поштовій папці;

19) Команда Close – закриває обрану поштову папку, після чого всі листи в ній будуть помічені прапорцем /deleted і фізично видаляються з неї;

20) Команда Expunge – видаляє з поштової папки листи, які позначені прапорцем /deleted;

21) Команда Search – виконує пошук листа у поштовій папці, який збігається з заданими у запиті критеріями.;

22) Команда Fetch – надсилає запит на отримання інформації про лист, це може бути дата, тіло, тема, розмір, його прапорці, унікальний ідентифікатор листа. Саме тому можна сказати що практично весь інтерфейс, який відображає нам звичний нам поштовий протокол в тому вигляді до якого ми звикли, побудований на цій команді;

23) Команда Store – змінює інформацію про лист у поштовій папці;

24) Команда Copy – копією повідомлення з однієї поштової папки в кінець іншої поштової папки;

25) Команда UID – це не команда поштового протоколу в звичному нам вигляді, а скоріше приставка, яка підставляється до будь-якої вище описаної команди і далі обрана команда буде виконувати ті самі дії тільки тепер буде працювати не зі звичайними ідентифікаторами листа, які завжди змінюються після певних дій, а з унікальними ідентифікаторами кожного листа, які завжди незмінні;

Тепер коли ми знаємо як працюють всі команди ми можемо розглянути команди що мають аргументи:

1) Команда AUTHENTICATE має аргумент – назву механізму автентифікації;

2) Команда LOGIN має аргумент – ім’я користувача та його пароль;

3) Команда SELECT має аргумент – назва поштової папки;

4) Команда EXAMINE має аргумент - назва поштової папки;

5) Команда CREATE має аргумент – назва поштової папки;

6) Команда DELETE має аргумент - назва поштової папки;

7) Команда RENAME має аргументи - назва існуючої поштової папки та нове ім’я поштової папки;

8) Команда SUBSCRIBE має аргумент – назва поштової папки;

9) Команда UNSUBSCRIBE має аргумент – назва поштової папки;

10) Команда LIST має аргумент – назва поштової папки з можливими символами;

11) Команда LSUB має аргумент – назва поштової папки з можливими символами;

12) Команда STATUS має аргументи – назва поштової папки та статус який ми шукаємо;

13) Команда APPEND має аргумент – назва поштової папки та (необов’язково) назви прапорців та (необов’язково) дата-час та тіло листа (літерал);

14) Команда SEARCH має аргументи – критерії пошуку (номер листа, ALL, ANSWERED, BCC <string>, BEFORE <date>, BODY <string>, CC <string>, DELETED, DRAFT, FLAGGED, FROM <string>, HEADER <field-name> <string>,

KEYWORD <flag>, LARGER <n>, NEW, NOT <search-key>, OLD, ON <date>, OR <search-key1> <search-key2>, RECENT, SEEN, SENTBEFORE <date>, SENTON <date>, SENTSINCE <date>, SINCE <date>, SMALLER<n>, SUBJECT <string>, TEXT <string>, TO <string>, UID <sequence set>, UNANSWERED, UNDELETED, UNDRAFT, UNFLAGGED, UNKEYWORD <flag>, UNSEEN) та (необов'язково) тип кодування тексту;

15) Команда FETCH має аргументи – номери листів, деталі листа які потрібно вивести;

16) Команда STORE має аргументи - номери листів, команда для зміни прапорця (FLAGS <flag list>, FLAGS.SILENT <flag list>, +FLAGS <flag list>, + FLAGS.SILENT <flag list>, -FLAGS <flag list>, - FLAGS.SILENT <flag list>);

17) Команда COPY має аргументи – номери листів, назва поштової папки;

18) Команда UID має аргументи – назва іншої команди, її аргумент;

А такі команди як, CAPABILITY, NOOP, LOGOUT, STARTTL, CHECK, CLOSE, EXPUNGE не мають аргументів, тому їхніми вхідними даними є тільки тег та власне команда.

Більш детально вважаю варто розповісти про літерал. В протоколі IMAP4 це основна форма передачі символів (тексту), які не несуть у собі команд та їх уточнень. Особливо це важливо для тексту в якому є CRLF (Carriage Return Line Feed) – символ що означає розрив чи перехід рядка. У протоколі IMAP4 введення такого формату тексту відбувається таким чином, що спочатку в команді яку ми хочемо виконати ми оголошуємо в фігурних дужках розмір нашого тексту - його кількість байтів. Потім сервер дає нам відповідь, що готовий отримувати дані формату літерал і далі сервер розглядає надану йому інформацію як звичайний блок, який йому не потрібно аналізувати.

1.2. Вихідні дані

Вихідними даними нашого проекту є відповіді сервера на наші команди.

Всі команди отримують однакову кінцеву відповідь. Це може бути OK, NO, BAD:

OK – означає, що команда успішно виконана і закінчена;

NO – означає, що відбулася операційна помилка сервера і команда не виконана;

BAD – означає, що команда некоректно написана і не може бути виконана.

Рідше ми зустрінемо такі відповіді сервера, як:

PREAUTH – одна з трьох можливих відповідей-вітань, означає, що вхід було виконано з автентифікацією і тому команда LOGIN не потрібна, використовується без тегу;

BYE – відповідь-прощання, означає що з'єднання з сервером завершене або розірване;

Також ці відповіді мають той самий тег, що і команда, на яку вони

відповідають та супроводжуються поясненнями від сервера.

У разі коли команда введена вірно сервер починає її виконувати і

надсилати інформацію на яку ми робили запит. У цьому разі, перед фінальною відповіддю про успішне виконання команди, сервер надсилає нам потрібну інформацію з позначкою * та назвою команди, яку виконує, на початку.

Наприклад: ми надсилаємо команду FETCH для того, щоб вона показала нам лист, його унікальний номер, розмір, дату, прапорці, тіло листа з темою, відправником. Команда буде виглядати так:

0010 FETCH 1 (UID RFC822.SIZE INTERNALDATE FLAGS

BODY[HEADER.FIELDS (SUBJECT FROM)])

Потім ми отримаємо таку відповідь від сервера:

* 1 FETCH (UID 1 RFC822.SIZE 11305 INTERNALDATE "27-Apr-2017 10:56:23 +0000" FLAGS (\Seen) BODY[HEADER.FIELDS (SUBJECT FROM)] {222}

< Subject: =?UTF-8?B?0KLRgNC4INC/0L7RgNCw0LTQuCDQtNC70Y8g0YPRgdC/0ZbRiNC90L7RlyD

RgNC+0LHQvg==?=

=?UTF-8?B?0YLQuCDQtyDQv9C+0YjRgtC+0Y4gR21haWw=?=

From: =?UTF-8?B?0JrQvtC80LDQvdC00LAgR21haWw=?= mail-noreply@google.com

0010 OK Success

На цьому прикладі ми наглядно бачимо в якому вигляді ми отримуємо відповідь від сервера на нашу команду.

Також ми можемо отримати відповідь з тегом +, що означає що після цієї відповіді сервер очікує на доповнення інформації для команди.

Наприклад коли ми за допомогою команди APPEND хочемо додати новий лист, що передбачає введення літералу, як тіла листа. Тому для того щоб виконати цю команду ми надсилаємо запит серверу:

A003 APPEND saved-messages (\Seen) {310}

Що означає що ми хочемо додати в папку saved-messages лист розміром 310 байтів з прапорцем Seen. І отримуємо відповідь:

+ Ready for literal data

Що означає, що після цього ми маємо ввести лист розміром 310 байтів як ми і вказали у запиті.

Після вводу отримуємо заключну відповідь про успішне завершення команди:

A003 OK APPEND completed

Для успішної роботи нашої бібліотеки їй потрібно вміти розрізняти та компонувати відповіді сервера правильно, відповідно до наших команд.

1.3 Кодування та декодування UTF7 IMAP4

Для повноцінної роботи нашого сервера нам перед тим як зрозуміти відповіді сервера потрібно їх розшифрувати, адже в наш час ніяка інформації не передається в мережі без мінімального рівня захисту шляхом кодування. В нашому продукті ми

використовуємо кодування UTF7(7-bit Unicode Transformation Format) для декодування тексту Юнікод. Це завжди виконується для текстів (не літералів) які написані кирилицею, або простіше говорячи не латинськими літерами.

Принцип кодування UTF7:

1) Спочатку перетворюємо Юнікод символи з hex-формату в бінарний формат;
2) Об’єднуємо бінарні послідовності;
3) Перегруповуємо двійковий код в блоки по 6 біт, починаючи зліва;
4) Якщо останній блок має менше 6 бітів, заповнюємо його нулями;
5) Заміняємо кожен блок відповідними кодами Base64;

І відповідно виконується декодування:

1) Кожен символ Base64 перетворюємо в бітову послідовність, яку він представляє;
2) Перегруповуємо двійковий код в групи по 16 бітів;
3) Якщо у кінці залишилася група яка повністю заповнена нулями, то її можна відкинути;
4) Кожну групу перетворюємо у відповідний номер символу Юнікоду;

1.4 Опис SSL (Security Socket Layer) протоколу та його важливість у протоколі IMAP4

Якщо ми вже доторкнулися до теми безпеки та шифрування у нашому проекті, то важливо також розповісти про з’єднання з сервером через SSL (Security Socket Layer).

SSL – це криптографічний протокол, який є стандартним для безпечного з’єднання схемою клієнт-сервер, а наш протокол IMAP4 використовує саме таке з’єднання. Завдяки цьому протоколу ми можемо конфіденційно обмінюватися даними між клієнтом та сервером. Цей протокол є важливою частиною моделі TCP/IP.

Для шифрування використовується асиметричний алгоритм з відкритим ключем. Для такого алгоритму використовують два ключі. Один з них використовують для шифрування, а другий відповідно для розшифрування. Один з ключів зберігається відкрито, другий навпаки зберігається в таємниці.

SSL надає канал для передачі даних і він має три властивості:

1) Аутентифікація – в той час як клієнт може бути не аутентифікований, залежно від алгоритму, то сервер завжди є аутентифікованим;

2) Цілісність – коли йде обмін даними по каналу, то обов'язково йде перевірка цілісності;

3) Конфіденційність каналу – SSL протокол починає свою роботу тільки після встановлення з'єднання, що забезпечує конфіденційність.

SSL протокол являє собою своєрідний фільтр, адже він розміщається по суті між двома протоколами:

1) Протокол який використовує програма-клієнт;

2) Транспортний протокол моделі TCP/IP;

Також роботу SSL протоколу можна розділити на два рівня:

1) Рівень протоколу для підключення з'єднання, його ще називають рукостискання (Handshake Protocol Layer);

2) Рівень протоколу запису;

Також прийнято вважати, що рівень протоколу рукостискання виконують одразу три інші протоколи:

1) Протокол підтвердження підключення(Handshake Protocol);

2) Протокол параметрів зміни шифру(Cipher Spec Protocol);

3) Попереджувальний протокол(Alert Protocol);

Для того, щоб підтвердити підключення використовуючи протокол підтвердження підключення і за допомогою нього виконується сесія підтвердження даних між клієнтом і сервером. Для цього потрібен стандартний набір даних сесії, а саме:

1) Ідентифікаційний номер сесії;

2) Сертифікати обох сторін – мається на увазі цифровий сертифікат, який підтверджує, що ви маєте право власності на відкритий ключ чи інші атрибути;

3) Параметри алгоритму шифрування;

4) Алгоритм стискання даних;

5) «Загальний секрет» який використовується для створення ключів; відкритий ключ;

Потім відбувається обмін цими даними, на основі чого відбувається аутентифікація сторін та вирішується шифрування, хешування та стискання. За все це відповідає протокол підтвердження підключення.

Для того щоб змінити дані ключа – інформації яка використовується для створення ключів використовується протокол параметрів зміни шифру. При чому цей протокол виконується лише однією командою, в якій сервер говорить, що він хоче змінити набір ключів.

Попереджувальний протокол містить повідомлення, яке показує всім сторонам зміну статусу або повідомляє про можливу помилку.

1.5 Висновки до розділу

Описані вище команди та відповіді сервера дають нам розуміння про роботу протоколу IMAP4, що стануть основою нашої бібліотеки методів. Методи будуть реалізовувати команди які будуть надсилатися протоколу і потім ми будемо одержувати відповіді сервера на наші запити. Ті символи що написані не латинськими літерами ми маємо закодувати або декодувати за допомогою UTF7 IMAP4.

2. Аналіз літератури та подібних рішень реалізації

Перш за все варто дослідити різницю протоколу IMAP4, та подібних популярних рішень.

2.1 Подібні поштові протоколи

Частіше за все протокол IMAP4 порівнюють з протоколом POP3. IMAP4 був розроблений, для того щоб замінити POP3, адже він був набагато простіший. Тому IMAP4 має велику кількість переваг проти POP3, а саме:

1) Листи зберігаються на сервері, а не на комп'ютері клієнта, що дає можливість отримувати доступ до поштової скриньки з різних пристроїв;
2) Багато можливостей для роботи з папками;
3) Можливість шифрування передачі пароля та безпечна передача трафіку за допомогою SSL;
4) Передбачає режим асинхронного взаємодії, що означає, що клієнт може направити сервера відразу кілька команд, не чекаючи відповідей на кожну з них, а потім прийняти разом всі відповіді. Для забезпечення такого режиму з кожною командою зв'язується унікальна мітка, яка дозволить сервера ідентифікувати команду, а клієнту – ідентифікувати відповідь;

2.2 Протоколи IMAP4

Також протокол IMAP пройшов довгий шлях у вигляді шести версій:

1) Original IMAP 1986;
2) IMAP2 1988, 1990;
3) IMAP3 1991;

4) IMAP2bis 1993;

5) IMAP4 1994;

6) IMAP4rev1 1996, 2003;

В нашому проекті було реалізовано протокол IMAP4rev1(IMAP, версія 4, ревізія 1), але саме його прийнято називати IMAP4.

2.3 Вибір мови розробки

Протокол IMAP4 досить широко використовується в нашому світі та моя бібліотека реалізує його саме на мові C++. Зараз на ринку є аналоги які реалізовані для інших мов програмування таких як PHP, Java, Python.

Всі вони по суті відрізняються тільки мовою реалізації, а всі основні функції, які повинні бути у протоколі IMAP4 однакові. Але що стосується легкості, компактності та швидкості коду, то на мою думку C++ значно виграє.

Тому хотілось би назвати переваги роботи з мовою програмування C++ для реалізації протоколу IMAP4:

1) Масштабування – C++ дає нам можливість реалізувати код який ми матимемо змогу підключити до різноманітних платформ, програм та систем;

2) Компактність – C++ дає можливість писати компактний код, коли з таким самим функціоналом в інший мові програмування программа може бути в два рази більша;

3) ООП (Об’єктно Орієнтованим Програмуванням) – C++ дає нам можливість роботи з ООП що є ідеальним для нашої бібліотеки, в якій ми створили клас, в якому і реалізували всі команди нашого протоколу;

Але не можна не згадати і про недоліки роботи з C++:

1) Надмірна кількість можливостей – це може призводити до виникнення помилок, які дуже важко відслідкувати;

2) Недостатня кількість даних про типи під час компіляції;

3) Наявність більш ніж одного механізму для виконання однієї задачі, що призводить до неоптимального кодування;

Для своєї роботи з цією мовою програмування була вивчена книга Герберта Шилдта “C++ Базовий курс”[1], яка допомогла мені в роботі з методами мого класу. Там наведено гарні приклади реалізації ООП в C++, які допомогли мені глибше розібратися зі специфікою роботи даного проекту.

2.4 Аналіз джерел вивчення протоколу

Починаючи досліджувати цю тему був проведений глибокий пошук джерел для детального вивчення протоколу IMAP4 та найкращим другом за цей час мені став документ RFC 3501 (Request For Comments)[2], що являє собою величезну технічну документацію про протокол з точними даними які повність описують роботу протоколу, кожну його команду, відповідь та усі тонкості, які траплялися на моєму шляху з реалізації всіх функцій коректної роботи протоколу IMAP4.

Також ця документація дає чітке визначення роботи та властивостей протоколу IMAP4. Це протокол доступу до інтернет повідомлень, які ми всі називаємо електронна пошта. Дозволяє клієнту отримувати доступ та керувати електронними поштовими повідомленнями на сервері. IMAP4 дозволяє маніпулювати поштовою скринькою та поштовими папками в ній.

IMAP4 включає операції зі створення, видалення та перейменування поштових папок, перевірка нових повідомлень, постійне видалення повідомлень, встановлення та очищення прапорців, розбір, пошук і вибіркове отримання атрибутів, текстів та частин повідомлення.

Повідомлення в IMAP4rev1 доступні за допомогою використання чисел. Ці номери є або послідовними номерами повідомлень, або унікальними ідентифікатори.

2.5 Допоміжні протоколи та модельTCP/IP

IMAP4 не вказує спосіб надсилання нових повідомлень електронної пошти. Ця функція виконується сторонніми поштовими протоколами для передачі пошти, таких як SMTP (Simple Mail Transfer Protocol).

SMTP – це текстовий синхронний протокол який складається з із серії команд. Хоча по суті цей протокол виконує тільки дві функції:

1) Перевіряє правильність налаштувань і видає дозвіл комп'ютера, який намагається відправити email-повідомлення.

2) Відправляє вихідне повідомлення на вказану адресу і засвідчується в успішну доставку повідомлення. Якщо його неможливо доставити, відправнику надсилається повідомлення про це.

Говорячи в попередніх підрозділах про схожі протоколи та тут про протокол відправки електронної пошти потрібно згадати що всі ці протоколи є частиною великої мережевої моделі TCP/IP.

TCP/IP - це систематизований стек протоколів мережі Інтернет. Являє собою модель, яка має чотири рівні. По своїй суті схожу на еталонну модель OSI. І хоч в OSI сім рівнів всі вони повністю функціонально реалізовані в моделі TCP/IP у чотирьох рівнях. Модель описує спосіб передачі даних з джерела інформації до отримувача.

Представляє рівневий підхід до мережі. Ділить взаємодію з мережею на рівні, які відповідають за певну дію та підхід до мережі. У моделі TCP/IP є чотири рівні:

1) Прикладний рівень;
2) Транспортний рівень;
3) Мережевий рівень;
4) Канальний рівень;

Прикладний рівень – протоколи цього рівня відповідають за взаємодію прикладних процесів та програм з мережею. Серед них вже знайомий нам IMAP4, POP3 та SMTP. Всі протоколи прикладного рівня мають призначення для

конкретних прикладних завдань.

Транспортний рівень – такі протоколи відповідають за надання транспортних процесів прикладним процесам. Кожен такий процес обов'язково взаємодію з якимось протоколом транспортного рівня частіше за все з TCP(Transmission Control Protocol) або UDP(User Datagram Protocol) через виділений порт. Порти нумерують з нуля. Тому при передачі запиту прикладної програми формують дейтаграму чи сегмент, в якому вказані номери портів програмних модулів клієнта та програмного модуля сервера. Саме тому в заголовку пакета протоколу транспортного рівня виділено два поля для номерів портів одержувача та відправника, які займають 2 байти.

Мережевий (міжмережевий) рівень – протоколи цього рівня відповідають за взаємодію різних архітектур. Основним протоколом мережевого рівня вважають IP(Internet Protocol). Його основна мета це маршрутизація пакетів даних між різними комп'ютерними мережами. Для цього міжмережевий протокол IP використовує IP-адресацію всіх мереж та вузлів. Тому зараз у нашому сучасному світі всі пристрої та комп'ютери мають унікальний числовий номер, який ми всі знаємо як IP-адреса. У IPv4 його довжина має 4 байти, але світ зараз намагається переходити на IPv6, а його довжина в свою чергу займає 16 байтів.

Канальний рівень – рівень доступу до середовища передачі. Описує спосіб кодування даних для передачі пакету даних на фізичному рівні.

Має дві функції, через що іноді канальний рівень розділяють на два рівні:

1) Перетворення та відображення IP-адреси в MAC-адресу. MAC-адреса - це фізична адреса мережі. Це перетворення відбувається за допомогою протоколу ARP(Address Resolution Protocol). Цікаво те що перетворення може відбуватися тільки під час відправлення IP-пакета, адже тільки в цей момент створюються заголовки IP та Ethernet. Потім ARP протокол формує таблицю і далі ми вже користуючись таблицею будемо шукати адресу і це і буде перетворення. Ця таблиця зберігається у пам'яті і містить дані про всі вузли у вигляді рядків таблиці.

2) Інкапсуляція IP-дейтаграм в кадри, щоб потім передавати їх по

фізичному каналу і передавати кадри.

2.5 Висновки до розділу

Ми дослідили подібні рішення протоколу з різних сторін. Тепер ми розуміємо які переваги дає нам протокол IMAP4 написаний у вигляді бібліотеки мовою C++. Проаналізували джерела інформації які допомагали нам при розробці проекту. Та повністю розібрали мережеву модель TCP/IP, яка є стеком протоколів для роботи з передачею даних у мережі. Частиною цієї моделі є протокол IMAP4. Також ми дізналися про інші допоміжні протоколи, що є дуже важливою частиною розуміння для роботи над нашим проектом, адже робота з електронною поштою це великий процес, який проходить через безліч протоколів та через всі рівні моделі TCP/IP.

3. Засоби розробки

3.1 Опис програмного середовища

Для розробки всього продукту було використане середовище Microsoft Visual Studio 2019.

Microsoft Visual Studio – це, по суті, серія продуктів від компанії Microsoft, що включають в себе інтегроване середовище розробки програмного забезпечення та багато інших інструментів, які допомагають в розробці програмних продуктів. Ця серія продуктів дозволяє розробляти консольні програми, програми з графічним інтерфейсом, з підтримкою Windows Forms, а також веб-сайти, веб-застосунки, веб-служби як в рідному, так і в керованому кодах для всіх платформ, що підтримуються Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, .NET Framework, .NET Compact Framework та Microsoft Silverlight[3].

Редактор вихідного коду з підтримкою технології IntelliSense та можливістю простого рефакторінгу коду включає в собі Visual Studio. Вже встановлений відладчик може працювати з рівнем вихідного коду та також і з машинним рівнем.

Крім того в Visual Studio є можливість підключати та встановлювати різноманітні плагіни та інструменти. Наприклад для спрощення створення графічного інтерфейсу додатку ми можемо встановити інструменти, які включають в себе редактор форм, веб-редактор та дизайнер схеми бази даних. При чому, що плагіни та сторонні доповнення можна встановлювати практично на кожному рівні. Наприклад додавання підтримки систем контролю версій вихідного коду, таких як Subversion та Visual SourceSafe . Додавання нових наборів інструментів, таких як інструменти для редагування і візуального проектування коду на об’єктно-орієнтованих мовах програмування, або таких як інструменти для інших аспектів процесу розробки програмного забезпечення, таких як клієнт Team Explorer для

роботи з Team Foundation Server.

Крім того не можу не відмітити інтерфейс та його зручність у середовищі Visual Studio.

В 2018 році компанія Microsoft зробила випуск оновленої версії Visual Studio, але тільки в статусі Preview. Наступним етапом випуску була версія випущена у березні зі статусом Release Candidate, потім ще через місяць була випущена стабільна версія Visual Studio 2019.

У новій версії дуже зручно реалізоване виявлення помилок та відладка коду. Мало того що середовище завжди підказує вам під час написання коду і може передбачити якісь ваші помилки, то тепер програма сама може виправляти ваші помилки. Вона виводить їх у вигляді списку в якому галочкою ми можемо виділити помилки, які середовище саме виправить. Також дуже зручним є те що для свого проекту ви можете самі встановити потрібні «гарячі» клавіші для швидких команд.

Всі ці інструменти можуть значно розширити можливості нашої роботи в Visual Studio. Саме тому я обрала цю програму для розробки своєї бібліотеки.

Оскільки вона написана мовою C++, то саме це середовище я вважаю найбільш зручним для розробки такого роду продуктів.

3.2 Мова розробки

Для розробки нашого продукту ми обрали мову програмування C++. Ця мова пройшла дуже великий шлях розвитку та її стандарти змінюються кожен період часу. Ця мова побудована на мові C. У новій версії C, а саме C++ була додана підтримка об’єктно-орієнтованого програмування через класи. Саме це нас і цікавить у ній для розробки нашого проекту, адже це бібліотека методів класу, що є фактично основою об’єктно-орієнтовного програмування.

Також мова C++ зараз є однією з найуживаніших мов програмування у світі, що дуже добре для нас, адже наша бібліотека не є фінальним продуктом, а має підключатися до інших систем чи програм. Тому нам потрібно щоб наш продукт був

універсальним та більш відкритим для широкої аудиторії.

3.2 Висновки до розділу

У даному розділі ми детально розглянули обране програмне середовище для розробки проекту бібліотеки методів для реалізації поштового протоколу IMAP4. Обраний засіб для розробки Microsoft Visual Studio 2019 повністю відповідає вимогам сучасного середовища, що дасть нам змогу створити продукт який буде відповідати всім стандартам сучасного користувача. Також ми дійшли висновку що мова програмування C++ є ідеальною для розробки нашого продукту, завдяки своїй поширеності та підтримці об’єктно-орієнтовного програмування.

4. Опис програмної реалізації

4.1. Програмна архітектура

Протокол IMAP4 працює з архітектурою клієнт-сервер. В цій архітектурі досить проста схема роботи, яка зображена на рисунку 4.1.

Модуль Клієнт – надсилає запит серверу у вигляді команди на надання певної інформації чи виконання дії в поштовій скриньці.

Модуль Сервер – отримує запит клієнта, надсилає відповідь про виконання запиту.

Рисунок 4.1 – схема роботи архітектури Клієнт-Сервер

У нашій бібліотеці реалізувано модуль Клієнта, який надсилає Серверу коректні запити, у вигляді стандартних команд протоколу IMAP4, отримує його відповіді і правильно їх зчитує, парсить та розшифровує.

Всі ці функції реалізовані в класі PlIMAP4Client у вигляді методів цього класу.

Опис цих методів надано в таблиці 4.2.

Таблиця 4.2 – методи класу PlIMAP4Client

Метод	Тип	Опис
Connect	void	Встановлення з’єдання з сервером.
Capability	void	Реалізує команду протоколу Capability. Запитує у сервера його список можливостей.

Метод	Тип	Опис
StartTLS	void	Реалізує команду протоколу StatTLS. Явно включає шифрування з’єднання.
Login	void	Реалізує команду протоколу Login. Ідентифікує користувача, надсилає ім’я та пароль користувача.
Logout	void	Реалізує команду протоколу Logout. Повідовляє сервер про те що користувач закінчив з’єднання.
KeepAlive	void	Реалізує команду протоколу Noop. Пуста команда, яка використовується для підтримання з’єднання з сервером.
List	void	Реалізує команду протоколу List. Надсилає запит на отримання списку всіх поштових папок клієнта.
Lsub	void	Реалізує команду протоколу Lsub. Надсилає запит на отримання списку всіх поштових папок клієнта, активізованих командою Subscribe.
Select	void	Реалізує команду протоколу Select. Обирає поштову папку, щоб мати доступ до її повідомлень.
Examine	void	Реалізує команду протоколу Examine. Ідентична до команди Select, проте обрана поштова скринька ідентифікується як лише для читання.
Status	void	Реалізує команду протоколу Status. Надсилає запит про поточний стан поштової

Метод	Тип	Опис
		папки.
Close	void	Реалізує команду протоколу Close. Закриває поштову папку, пясля чого всі листи в ній будуть помічені прапорцем /deleted і фізично видаляються з неї.
Fetch	void	Реалізує команду протоколу Fetch. Надсилає запит на отримання інформації про лист.
Store	void	Реалізує команду протоколу Store. Змінює інформацію про лист.
Append	void	Реалізує команду протоколу Append. Додає повідомлення в кінець вказаної поштової папки.
Copy	void	Реалізує команду протоколу Copy. Копіює повідомлення з однієї поштової папки в іншу.
Expunge	void	Реалізує команду протоколу Expunge. Видаляє з поштової папки листи які позначені прапорцем /deleted.
Create	void	Реалізує команду протоколу Create. Створює нову поштову папку в нашій поштовій скринці.
Rename	void	Реалізує команду протоколу Rename. Змінює ім’я поштової папки.
Delete	void	Реалізує команду протоколу Delete. Видаляє поштову папку та всі повідомлення в ній.

Метод	Тип	Опис
Check	void	Реалізує команду протоколу Check. Встановлює контрольну точку у вказаній поштовій папці.
Subscribe	void	Реалізує команду протоколу Subscribe. Додає вказану поштову папку в список активних папок клієнта.
Unsubscribe	void	Реалізує команду протоколу Unsubscribe. Видаляє вказану поштову папку зі списку активних папок клієнта.

Практично всі методи описаного класу є реалізацією стандартних команд клієнта протоколу IMAP4, тому їх аргументи описані у розділі 1.1 є параметрами відповідних команді методів.

4.2. Графічний опис системи

При створенні програмного продукту необхідно чітко розуміти його архітектуру і властивості. Для опису архітектури і властивостей програмних систем використовуються різні методи. Серед них є графічний метод. Візуалізація даних у графічній формі сприяє їх сприйняттю.

Зокрема для цілей графічного представлення програмних систем використовуються мова графічного опису для моделювання програмних систем Unified Modelling Language(UML) [4].

Ця технологія використовує діаграми різних типів. Серед них:

– діаграма компонентів – відображає залежності між компонентами програмного забезпечення;

– діаграма станів – діаграма що відображає стани об’єкта у часі, показує його перехід з одного стану в інший, ця діаграма служить для моделювання динамічних аспектів програми, однак в нашому випадку ми описуємо стани які вже передбачені самим протоколом;

4.2.1 Діаграма компонентів бібліотеки IMAP4 клієнта.

У наведеній нижче діаграмі компонентів (рисунок 4.2.1) ми можемо розглянути взаємодію компонентів нашої бібліотеки, які реалізують коректну роботу протоколу на всіх рівнях з’єднання.

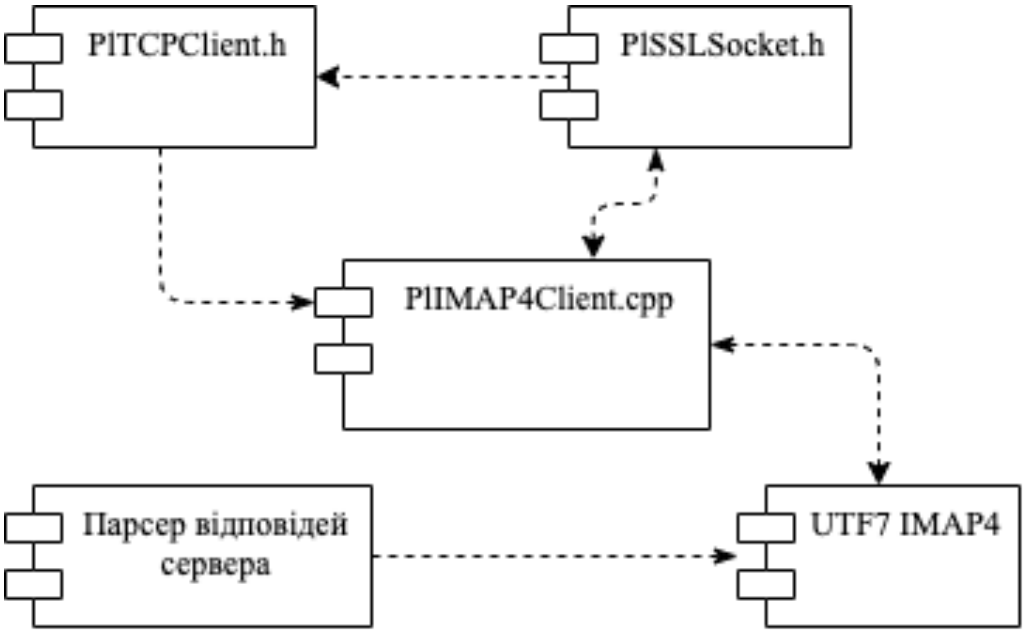

Рисунок 4.2.1 – діаграма компонентів

1) PlTCPClient.h – відповідає за з’єднання з потрібним сокетом;

2) PlSSLSocket.h – відповідає за безпечне з’єднання з потрібним сокетом на рівні SSL(Secure Sockets Layer);

3) Парсер відповідей сервера – відповідає за обробку відповідей сервера і формування їх у логічні блоки;

4) UTF 7 IMAP4 – відповідає за декодування отриманих та сформованих відповідей сервера

5) PlIMAP4Client.cpp – наша бібліотека, яка з усіма під'єднаними компонентами реалізує коректну роботу протоколу IMAP4;

За допомогою даної діаграми ми бачимо, як компоненти нашого проекту взаємодіють між собою для коректного виконання протоколу IMAP4.

4.2.2 Діаграма станів IMAP4 клієнта.

Розглянемо детально перехід станів нашого клієнта під час роботи з сервером у протоколі IMAP4 на діаграмі станів (рисунок 4.2.2).

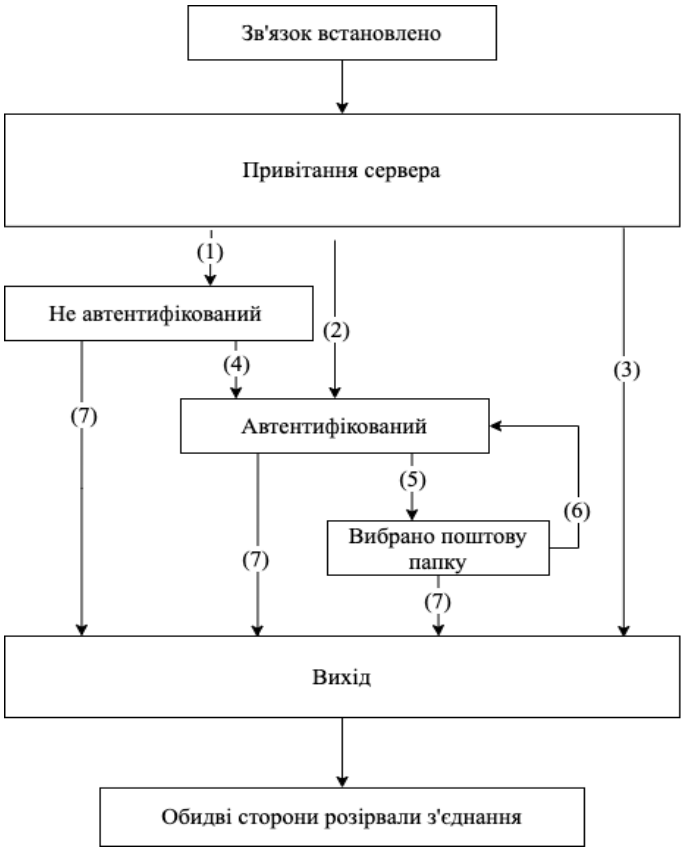

Рисунок 4.2.2 Діаграма станів

Стрілочками показано, яка дія відбулася для переходу одного стану в інший. Опис дій:

1) З'єднання без попередньої автентифікації (OK вітання);
2) З'єднання з попередньою автентифікацією (PREAUGTH вітання);
3) Відкинуте з'єднання (BYE вітання);
4) Успішні команди LOGIN або AUTHENTICATE;

5) Успішні команди SELECT або EXAMINE;

6) Команда CLOSE, або помилкові команди SELECT або EXAMINE;

7) Команда LOGOUT, сервер завершив роботу, або з’єднання розірвано;

За допомогою цих графічних переходів, ми бачимо як працює наш протокол зсередини. Це сприяє кращому розумінню нашої системи перед тим як ми починаємо її розробляти, що дуже важливо для коректної роботи програми з протоколом IMAP4.

4.3 Висновки до розділу

У даному розділі ми розглянули програмну архітектуру нашого проекту у вигляді основних методів класу, які реалізують основні команди протоколу IMAP4. Також для кращого розуміння роботи протоколу та тонкощів його станів та компонентів ми розглянули діаграми станів та компонентів, які наглядно показують нам роботу протоколу та нашої реалізації протоколу в проекті.

5. Робота користувача з програмною системою

5.1 Вимоги до системи для інсталяції

Розроблена бібліотека методів є лише реалізацією роботи протоколу IMAP4. Задумана вона, як частина програми, яка реалізує роботу з електронною поштою. Тому кінцевим користувачем проекту є програміст, який буде підключати цю бібліотеку до свого проекту. Вимог до підключення є всього три:

1) Бібліотека працює лише на базі операційної системи Windows;

2) Бібліотека написана мовою C++, тому проект до якого вона буде підключатися також має бути написаний мовою C++;

3) Середовище розробки Microsoft Visual Studio;

За дотримання цих умов бібліотека буде працювати як окремий модуль системи. Він буде реалізовувати поштовий модуль, який можна використовувати як:

1) Частину email клієнта;

2) Реалізація взаємодії бізнес-систем з електронною поштою таких як CRM (Client Relationship Management);

3) Агрегація відслідковування новин, які розповсюджуються через email;

Завдяки цьому модулю можна значно розширити функціонал системи, налагодити комунікацію у ній та і просто мати у функціоналі модуль який коректно працює з поштою та відповідає всім запитам сучасного користувача зі своїми поштовими скриньками. Все це реалізовано саме в даному проекті описаними вище методами.

Також пропоную переглянути демонстрацію проекту реалізації поштового протоколу IMAP4 для того, щоб наглядно пересвідчитися у коректності роботи бібліотеки методів з поштовими запитами клієнта протоколу IMAP4. Саме це послужить найкращим доказом цінності мого проекту.

5.2 Демонстрація роботи проекту

У якості демонстрації була розроблена консоль, яка показує результати роботи бібліотеки. В ній продемонстрована робота всіх методів класу, які відповідають стандартним командам протоколу IMAP4.

Також для демонстрації свого проекту було створено електронну пошту для роботи з нею, за допомогою створеної бібліотеки. Для цього було використано сервіс Gmail.

Gmail – це безкоштовний сервіс електронної пошти. В ньому можна надати доступ до вашої поштової скриньки іншим програмам. Для них він генерує пароль, який ми вводимо аргументом в команду Login.

Далі наведено приклади, у вигляді скриншотів (рисунки 5.1 – 5.26), роботи деяких команд у вигляді того як вони реалізовані в коді та виконання їх у консолі.

```
CPlIMAP4Client client;
CPlIMAP4MailBoxesList *boxes = new CPlIMAP4MailBoxesList();
CPlIMAP4MailBoxState mailBoxState;
CPlIMAP4FetchResponse *fetchResponse = new CPlIMAP4FetchResponse();

CPlString userName("pliasimap4@gmail.com");
CPlString password("eupacauitorbvnmr");
CPlString host("imap.gmail.com");

_setmode(_fileno(stdout), _O_U16TEXT);

AddMultiEventHandler(&(client.OnLog), &g_Log, CIMAPEvents::OnIMAP4Log);
client.SetUserName(userName);
client.SetPassword(password);
client.SetSSLPassThrough(FALSE);
client.Connect(host, 993);
client.Login();
```

Рисунок 5.1 – реалізація коду для команд Connect та Login

```
S: * OK Gimap ready for requests from 185.223.114.53 s25mb150590106ljs
C: 0 CAPABILITY
S: * CAPABILITY IMAP4rev1 UNSELECT IDLE NAMESPACE QUOTA ID XLIST CHILDREN X-GM-EXT-1 XYZZY SASL-IR AUTH=XOAUTH2 AUTH=PLA
IN AUTH=PLAIN-CLIENTTOKEN AUTH=OAUTHBEARER AUTH=XOAUTH
S: 0 OK Thats all she wrote! s25mb150590106ljs
C: 1 LOGIN ******** ********
S: * CAPABILITY IMAP4rev1 UNSELECT IDLE NAMESPACE QUOTA ID XLIST CHILDREN X-GM-EXT-1 UIDPLUS COMPRESS=DEFLATE ENABLE MOV
E CONDSTORE ESEARCH UTF8=ACCEPT LIST-EXTENDED LIST-STATUS LITERAL- SPECIAL-USE APPENDLIMIT=35651584
S: 1 OK pliasimap4@gmail.com authenticated (Success)
```

Рисунок 5.2 – демонстрація роботи команд Connect та Login в консолі

```
CPlWideString mailBox(L"");
CPlWideString refName(L"*");
client.List(mailBox, refName, &boxes);

wprintf(L"------------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"------------------------------------------------------------------------------\n");
```

Рисунок 5.3 – реалізація коду для команди List

```
C: 2 LIST "" *
S: * LIST (\HasNoChildren) "/" "INBOX"
S: * LIST (\HasChildren \Noselect) "/" "[Gmail]"
S: * LIST (\Flagged \HasNoChildren) "/" "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-"
S: * LIST (\HasNoChildren \Important) "/" "[Gmail]/&BBIEMAQ2BDsEOAQyBD4-"
S: * LIST (\HasNoChildren \Trash) "/" "[Gmail]/&BBoEPgRIBDgEOg-"
S: * LIST (\HasNoChildren \Sent) "/" "[Gmail]/&BB0EMAQ0BFYEQQQ7BDAEPQRW-"
S: * LIST (\HasNoChildren \Junk) "/" "[Gmail]/&BCEEPwQwBDw-"
S: * LIST (\All \HasNoChildren) "/" "[Gmail]/&BCMEQQRP- &BD8EPgRIBEIEMA-"
S: * LIST (\Drafts \HasNoChildren) "/" "[Gmail]/&BCcENQRABD0ENQRCBDoEOA-"
S: 2 OK Success
------------------------------------------------------------------------------
Folders:
INBOX
[Gmail]
[Gmail]/Із зірочкою
[Gmail]/Важливо
[Gmail]/Кошик
[Gmail]/Надіслані
[Gmail]/Спам
[Gmail]/Уся пошта
[Gmail]/Чернетки
------------------------------------------------------------------------------
```

Рисунок 5.4 – демонстрація роботи команди List в консолі

```
client.Examine(boxes->Get(2)->strName, &mailBoxState);

wprintf(L"------------------------------------------------------------------------------\n");
wprintf(L"Selected folder (for read only) %s\n", (wchar_t*)client.GetCurrentMailbox());
wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
wprintf(L"------------------------------------------------------------------------------\n");
```

Рисунок 5.5 – реалізація коду для команди Examine

```
C: 3 EXAMINE "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-"
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing)
S: * OK [PERMANENTFLAGS ()] Flags permitted.
S: * OK [UIDVALIDITY 4] UIDs valid.
S: * 5 EXISTS
S: * 0 RECENT
S: * OK [UIDNEXT 8] Predicted next UID.
S: * OK [HIGHESTMODSEQ 3871]
S: 3 OK [READ-ONLY] [Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg- selected. (Success)
--------------------------------------------------------------------------
Selected folder (for read only) [Gmail]/Із зірочкою
Total messages 5
Recent messages 0
--------------------------------------------------------------------------
```

Рисунок 5.6 – демонстрація роботи команди Examine в консолі

```
client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);
```

Рисунок 5.7 – реалізація коду для команди Fetch

```
--------------------------------------------------------------------------
C: 4 FETCH 1:* (RFC822.SIZE BODY[HEADER.FIELDS (SUBJECT)] INTERNALDATE FLAGS UID)
S: * 1 FETCH (UID 1 RFC822.SIZE 4737 INTERNALDATE "11-May-2020 11:20:13 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIELD
 (SUBJECT)] {18}
S: Subject: imap4


S: )
S: * 2 FETCH (UID 2 RFC822.SIZE 4744 INTERNALDATE "11-May-2020 11:18:59 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIELD
 (SUBJECT)] {30}
S: Subject: internet protocol


S: )
S: * 3 FETCH (UID 5 RFC822.SIZE 5080 INTERNALDATE "11-May-2020 11:15:28 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIELD
 (SUBJECT)] {18}
S: Subject: imap4


S: )
S: * 4 FETCH (UID 6 RFC822.SIZE 4821 INTERNALDATE "11-May-2020 14:33:44 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIELD
 (SUBJECT)] {20}
S: Subject: cooking


S: )
S: * 5 FETCH (UID 7 RFC822.SIZE 4737 INTERNALDATE "11-May-2020 14:33:05 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIELD
 (SUBJECT)] {19}
S: Subject: School


S: )
S: 4 OK Success
--------------------------------------------------------------------------
Message 0 size: 4737 subject: Subject: imap4
Message 1 size: 4744 subject: Subject: internet protocol
Message 2 size: 5080 subject: Subject: imap4
Message 3 size: 4821 subject: Subject: cooking
Message 4 size: 4737 subject: Subject: School
--------------------------------------------------------------------------
```

Рисунок 5.8 – демонстрація роботи команди Fetch в консолі

```
client.Select(boxes->Get(2)->strName, &mailBoxState);

wprintf(L"-------------------------------------------------------------------------------\n");
wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
wprintf(L"-------------------------------------------------------------------------------\n");
```

Рисунок 5.9 – реалізація коду для команди Select

```
--------------------------------------------------------------------------
C: 5 SELECT "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-"
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing \*)] Flags permitted.
S: * OK [UIDVALIDITY 4] UIDs valid.
S: * 5 EXISTS
S: * 0 RECENT
S: * OK [UIDNEXT 8] Predicted next UID.
S: * OK [HIGHESTMODSEQ 10626]
S: 5 OK [READ-WRITE] [Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg- selected. (Success)
--------------------------------------------------------------------------
Selected folder [Gmail]/Із зірочкою
Total messages 5
Recent messages 0
--------------------------------------------------------------------------
```

Рисунок 5.10 – демонстрація роботи команди Select в консолі

```
client.Store(imapRemove, imapSeenFlag, true, 0, imapLastMessage, false, NULL);
```

Рисунок 5.11 – реалізація коду для команди Store

```
--------------------------------------------------------------------------
C: 7 STORE 1:* -FLAGS.SILENT (\Seen)
S: 7 OK Success
```

Рисунок 5.12 – демонстрація роботи команди Store в консолі

```
client.Status(boxes->Get(2)->strName, imapMessages | imapUnseen | imapRecent, &mailBoxState);

wprintf(L"-------------------------------------------------------------------------------\n");
wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
wprintf(L"Unseen messages %i\n", mailBoxState.dwUnseenMessages);
wprintf(L"-------------------------------------------------------------------------------\n");
```

Рисунок 5.13 – реалізація коду для команди Status

```
C: 8 STATUS "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-" (MESSAGES UNSEEN RECENT)
S: * STATUS "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-" (MESSAGES 5 RECENT 0 UNSEEN 5)
S: * 1 FETCH (UID 1 FLAGS (\Flagged))
S: * 2 FETCH (UID 2 FLAGS (\Flagged))
S: * 3 FETCH (UID 5 FLAGS (\Flagged))
S: * 4 FETCH (UID 6 FLAGS (\Flagged))
S: * 5 FETCH (UID 7 FLAGS (\Flagged))
S: 8 OK Success
--------------------------------------------------------------------------
Selected folder [Gmail]/Із зірочкою
Total messages 5
Recent messages 0
Unseen messages 5
--------------------------------------------------------------------------
```

Рисунок 5.14 – демонстрація роботи команди Status в консолі

```
client.Create(mailBoxName);

client.List(mailBox, refName, &boxes);
wprintf(L"--------------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"--------------------------------------------------------------------------------\n");
```

Рисунок 5.15 – реалізація коду для команди Create

```
--------------------------------------------------------------------------
C: 11 CREATE "[Temporary]"
S: 11 OK Success
C: 12 LIST "" *
S: * LIST (\HasNoChildren) "/" "INBOX"
S: * LIST (\HasChildren \Noselect) "/" "[Gmail]"
S: * LIST (\Flagged \HasNoChildren) "/" "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-"
S: * LIST (\HasNoChildren \Important) "/" "[Gmail]/&BBIEMAQ2BDsEOAQyBD4-"
S: * LIST (\HasNoChildren \Trash) "/" "[Gmail]/&BBoEPgRIBDgEOg-"
S: * LIST (\HasNoChildren \Sent) "/" "[Gmail]/&BB0EMAQ0BFYEQQQ7BDAEPQRW-"
S: * LIST (\HasNoChildren \Junk) "/" "[Gmail]/&BCEEPwQwBDw-"
S: * LIST (\All \HasNoChildren) "/" "[Gmail]/&BCMEQQRP- &BD8EPgRIBEIEMA-"
S: * LIST (\Drafts \HasNoChildren) "/" "[Gmail]/&BCcENQRABD0ENQRCBDoEOA-"
S: * LIST (\HasNoChildren) "/" "[Temporary]"
S: 12 OK Success
--------------------------------------------------------------------------
Folders:
INBOX
[Gmail]
[Gmail]/Із зірочкою
[Gmail]/Важливо
[Gmail]/Кошик
[Gmail]/Надіслані
[Gmail]/Спам
[Gmail]/Уся пошта
[Gmail]/Чернетки
[Temporary]
--------------------------------------------------------------------------
```

Рисунок 5.16 – демонстрація роботи команди Create в консолі

Для демонстрації роботи команди Create ми також викликаємо та виконуємо

команду List.

```
client.Copy(mailBoxName, 1, 2, false, NULL);

client.Select(mailBoxName, &mailBoxState);

wprintf(L"-------------------------------------------------------------------------\n");
wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
wprintf(L"-------------------------------------------------------------------------\n");

client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);

wprintf(L"-------------------------------------------------------------------------\n");
for (int i = 0; i < fetchResponse->Count(); i++)
{
    CPlString messageSubject;
    fetchResponse->Get(i)->GetByName("BODY[HEADER.FIELDS (SUBJECT)]")->GetAsString(messageSubject);
    messageSubject.TrimRight();
    wprintf(L"Message %i size: %i subject: %S\n", i, fetchResponse->Get(i)->GetByName("RFC822.SIZE")->GetAsNumber(),
        (char*)messageSubject);
}
wprintf(L"-------------------------------------------------------------------------\n");
```

Рисунок 5.17 – реалізація коду для команди Copy

```
--------------------------------------------------------------------------
C: 13 COPY 1:2 "[Temporary]"
S: 13 OK [COPYUID 36 1:2 1:2] (Success)
C: 14 SELECT "[Temporary]"
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing \*)] Flags permitted.
S: * OK [UIDVALIDITY 36] UIDs valid.
S: * 2 EXISTS
S: * 0 RECENT
S: * OK [UIDNEXT 3] Predicted next UID.
S: * OK [HIGHESTMODSEQ 10736]
S: 14 OK [READ-WRITE] [Temporary] selected. (Success)
--------------------------------------------------------------------------
Selected folder [Temporary]
Total messages 2
Recent messages 0
--------------------------------------------------------------------------
C: 15 FETCH 1:* (RFC822.SIZE BODY[HEADER.FIELDS (SUBJECT)] INTERNALDATE FLAGS UID)
S: * 1 FETCH (UID 1 RFC822.SIZE 4737 INTERNALDATE "11-May-2020 11:20:13 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIE
 (SUBJECT)] {18}
S: Subject: imap4


S: )
S: * 2 FETCH (UID 2 RFC822.SIZE 4744 INTERNALDATE "11-May-2020 11:18:59 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIE
 (SUBJECT)] {30}
S: Subject: internet protocol


S: )
S: 15 OK Success
--------------------------------------------------------------------------
Message 0 size: 4737 subject: Subject: imap4
Message 1 size: 4744 subject: Subject: internet protocol
--------------------------------------------------------------------------
```

Рисунок 5.18 – демонстрація роботи команди Copy

Для демонстрації роботи команди Copy також викликаємо та виконуємо команди Select та Fetch.

```
client.Rename(mailBoxName, mailBoxNewName);

client.List(mailBox, refName, &boxes);
wprintf(L"-------------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"-------------------------------------------------------------------------------\n");
```

Рисунок 5.19 – реалізація коду для команди Rename

```
--------------------------------------------------------------------------
C: 16 RENAME "[Temporary]" "[Temporary New]"
S: 16 OK Success
C: 17 LIST "" *
S: * LIST (\HasNoChildren) "/" "INBOX"
S: * LIST (\HasChildren \Noselect) "/" "[Gmail]"
S: * LIST (\Flagged \HasNoChildren) "/" "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-"
S: * LIST (\HasNoChildren \Important) "/" "[Gmail]/&BBIEMAQ2BDsEOAQyBD4-"
S: * LIST (\HasNoChildren \Trash) "/" "[Gmail]/&BBoEPgRIBDgEOg-"
S: * LIST (\HasNoChildren \Sent) "/" "[Gmail]/&BB0EMAQ0BFYEQQQ7BDAEPQRW-"
S: * LIST (\HasNoChildren \Junk) "/" "[Gmail]/&BCEEPwQwBDw-"
S: * LIST (\All \HasNoChildren) "/" "[Gmail]/&BCMEQQRP- &BD8EPgRIBEIEMA-"
S: * LIST (\Drafts \HasNoChildren) "/" "[Gmail]/&BCcENQRABD0ENQRCBDoEOA-"
S: * LIST (\HasNoChildren) "/" "[Temporary New]"
S: 17 OK Success
--------------------------------------------------------------------------
Folders:
INBOX
[Gmail]
[Gmail]/Із зірочкою
[Gmail]/Важливо
[Gmail]/Кошик
[Gmail]/Надіслані
[Gmail]/Спам
[Gmail]/Уся пошта
[Gmail]/Чернетки
[Temporary New]
--------------------------------------------------------------------------
```

Рисунок 5.20 – демонстрація роботи команди Rename в консолі

Для демонстрації роботи команди Rename також викликаємо та виконуємо команду List.

```
client.Select(mailBoxNewName, &mailBoxState);

PlDateTime dateTime = 0;
CPlFileStream stream;
stream.Open((TCHAR*)L"1.eml", false, true, false, true, true);
client.Append(mailBoxNewName, 0, dateTime, false, &stream);
stream.Close();

client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);

wprintf(L"-------------------------------------------------------------------------------\n");
for (int i = 0; i < fetchResponse->Count(); i++)
{
    CPlString messageSubject;
    fetchResponse->Get(i)->GetByName("BODY[HEADER.FIELDS (SUBJECT)]")->GetAsString(messageSubject);
    messageSubject.TrimRight();
    wprintf(L"Message %i size: %i subject: %S\n", i, fetchResponse->Get(i)->GetByName("RFC822.SIZE")->GetAsNumber(),
        (char*)messageSubject);
}
wprintf(L"-------------------------------------------------------------------------------\n");
```

Рисунок 5.21 – реалізація коду для команди Append

```
------------------------------------------------------------------------
C: 18 SELECT "[Temporary New]"
S: * FLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing)
S: * OK [PERMANENTFLAGS (\Answered \Flagged \Draft \Deleted \Seen $NotPhishing $Phishing \*)] Flags permitted.
S: * OK [UIDVALIDITY 36] UIDs valid.
S: * 2 EXISTS
S: * 0 RECENT
S: * OK [UIDNEXT 3] Predicted next UID.
S: * OK [HIGHESTMODSEQ 10752]
S: 18 OK [READ-WRITE] [Temporary New] selected. (Success)
C: 19 APPEND "[Temporary New]" {73708}
S: + go ahead
S: * 3 EXISTS
S: 19 OK [APPENDUID 36 3] (Success)
C: 20 FETCH 1:* (RFC822.SIZE BODY[HEADER.FIELDS (SUBJECT)] INTERNALDATE FLAGS UID)
S: * 1 FETCH (UID 1 RFC822.SIZE 4737 INTERNALDATE "11-May-2020 11:20:13 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIE
 (SUBJECT)] {18}
S: Subject: imap4


S: )
S: * 2 FETCH (UID 2 RFC822.SIZE 4744 INTERNALDATE "11-May-2020 11:18:59 +0000" FLAGS (\Flagged \Seen) BODY[HEADER.FIE
 (SUBJECT)] {30}
S: Subject: internet protocol


S: )
S: * 3 FETCH (UID 3 RFC822.SIZE 73708 INTERNALDATE "02-Jun-2020 11:10:07 +0000" FLAGS (\Seen) BODY[HEADER.FIELDS (SUB
T)] {181}
S: Subject: =?UTF-8?Q?=5B=F0=9F=91=BC=20Happy=20Children=27s=20Day=F0=9F=8E=88Get=20?=
 =?UTF-8?Q?=24=30=2E=30=31=20Gift=5D=20Xiaomi=20Redmi=20=31=30X=20Pre-orde?=
 =?UTF-8?Q?r?=


S: )
S: 20 OK Success
------------------------------------------------------------------------
Message 0 size: 4737 subject: Subject: imap4
Message 1 size: 4744 subject: Subject: internet protocol
Message 2 size: 73708 subject: Subject: =?UTF-8?Q?=5B=F0=9F=91=BC=20Happy=20Children=27s=20Day=F0=9F=8E=88Get=20?=
 =?UTF-8?Q?=24=30=2E=30=31=20Gift=5D=20Xiaomi=20Redmi=20=31=30X=20Pre-orde?=
 =?UTF-8?Q?r?=
------------------------------------------------------------------------
```

Рисунок 5.22 – демонстрація роботи команди Append в консолі

Для демонстрації роботи команди Append спочатку виконуємо команду Select, щоб обрати поштову папку в яку ми додамо лист, потім зчитуємо файл, який ми будемо додавати, тільки потім викликаємо команду Append і для наглядного прикладу викликаємо команду Fetch.

```
client.Delete(mailBoxNewName);

client.List(mailBox, refName, &boxes);
wprintf(L"------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"------------------------------------------------------------------------\n");
```

Рисунок 5.23 – реалізація коду для команди Delete

```
C: 21 DELETE "[Temporary New]"
S: 21 OK Success
C: 22 LIST "" *
S: * LIST (\HasNoChildren) "/" "INBOX"
S: * LIST (\HasChildren \Noselect) "/" "[Gmail]"
S: * LIST (\Flagged \HasNoChildren) "/" "[Gmail]/&BAYENw- &BDcEVgRABD4ERwQ6BD4ETg-"
S: * LIST (\HasNoChildren \Important) "/" "[Gmail]/&BBIEMAQ2BDsEOAQyBD4-"
S: * LIST (\HasNoChildren \Trash) "/" "[Gmail]/&BBoEPgRIBDgEOg-"
S: * LIST (\HasNoChildren \Sent) "/" "[Gmail]/&BB0EMAQ0BFYEQQQ7BDAEPQRW-"
S: * LIST (\HasNoChildren \Junk) "/" "[Gmail]/&BCEEPwQwBDw-"
S: * LIST (\All \HasNoChildren) "/" "[Gmail]/&BCMEQQRP- &BD8EPgRIBEIEMA-"
S: * LIST (\Drafts \HasNoChildren) "/" "[Gmail]/&BCcENQRABD0ENQRCBDoEOA-"
S: 22 OK Success
--------------------------------------------------------------------------------
Folders:
INBOX
[Gmail]
[Gmail]/Із зірочкою
[Gmail]/Важливо
[Gmail]/Кошик
[Gmail]/Надіслані
[Gmail]/Спам
[Gmail]/Уся пошта
[Gmail]/Чернетки
--------------------------------------------------------------------------------
```

Рисунок 5.24 – демонстрація роботи команди Delete в консолі

Для демонстрації роботи команди Delete також викликаємо та виконуємо команду List.

```
client.Logout();
```

Рисунок 5.25 – реалізація коду для команди Logout

```
C: 12 LOGOUT
S: * BYE LOGOUT Requested
```

Рисунок 5.26 – демонстрація роботи команди Logout в консолі

На даних рисунках ми бачимо як рядки коду, які виконують різні стандартні команди протоколу IMAP4, отримують параметри відповідні стандартним аргументам для відповідних команд та в самій консолі ми бачимо стандартний вивід протоколу IMAP4, в якому чітко показана архітектура клієнт-сервер. Також ми бачимо ті базові команди які найчастіше використовуються для перегляду результату інших це – Fetch, List та Select.

Саме ці результати найкраще показують коректну роботу нашого проекту.

5.3 Демонстрація різних методів підключення

У випадку протоколу IMAP4 ми маємо тільки два типи з’єднання клієнта з сервером:

1) Безпечне;
2) Не безпечне;

Приклад безпечного з’єднання ми продемонстрували у попередньому підрозділі на рисунку 5.1-5.2. Там ми використали метод SetSSLPassThorought в який передали параметр false. Саме він відповідає за безпечне з’єднання.

На сьогодні зробите не безпечне з’єднання в інтернеті практично неможливо. Якщо ви спробуєте зробити таке з’єднання з сервером Google Gmail, то нічого просто не станеться, закінчить час з’єднання. Тому для того щоб продемонструвати в своїй роботі з’єднання не безпечне і потім зробити його безпечним завдяки методу StartTLS мені довелося встановлювати на своєму комп’ютері власний локальний поштовий сервер hMailServer. Також довелося згенерувати сертифікат та закритий ключ для перетворення з’єднання з не безпечного в безпечне.

Демонстрацію роботи не безпечного з’єднання з локальним поштовим сервером ми можемо переглянути на рисунках 5.27 та 5.28.

```
CPlString localUserName("olha@localhost");
CPlString localPassword("1808");
CPlString localHost("localhost");

client.SetUserName(localUserName);
client.SetPassword(localPassword);
client.Connect(localHost, 143);
client.Login();

client.List(mailBox, refName, &boxes);

wprintf(L"-------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"-------------------------------------------------------------------------\n");

client.Logout();
```

Рисунок 5.27 – реалізація коду для не безпечного з’єднання протоколу IMAP4

```
S: * OK IMAPrev1
C: 0 CAPABILITY
S: * CAPABILITY IMAP4 IMAP4rev1 CHILDREN IDLE QUOTA SORT ACL STARTTLS NAMESPACE RIGHTS=texk
S: 0 OK CAPABILITY completed
C: 1 LOGIN ******** ********
S: 1 OK LOGIN completed
C: 2 CAPABILITY
S: * CAPABILITY IMAP4 IMAP4rev1 CHILDREN IDLE QUOTA SORT ACL STARTTLS NAMESPACE RIGHTS=texk
S: 2 OK CAPABILITY completed
C: 3 LIST "" *
S: * LIST (\HasNoChildren) "." "INBOX"
S: 3 OK LIST completed
--------------------------------------------------------------------------------
Folders:
INBOX
--------------------------------------------------------------------------------
C: 4 LOGOUT
S: * BYE Have a nice day
```

Рисунок 5.28 – демонстрація роботи не безпечного з’єднання протоколу IMAP4 в консолі

Демонстрацію роботи протоколу IMAP4 з не безпечним з’єднанням та демонстрацію роботи методу StartTLS для явного вмикання шифрування та відповідно безпечного з’єднання ми можемо переглянути на рисунках 5.29 та 5.30.

```
client.Connect(localHost, 143);
client.Login();
client.StartTLS();

client.List(mailBox, refName, &boxes);

wprintf(L"--------------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"--------------------------------------------------------------------------------\n");

client.Logout();
```

Рисунок 5.29 – реалізація коду для команди StartTLS

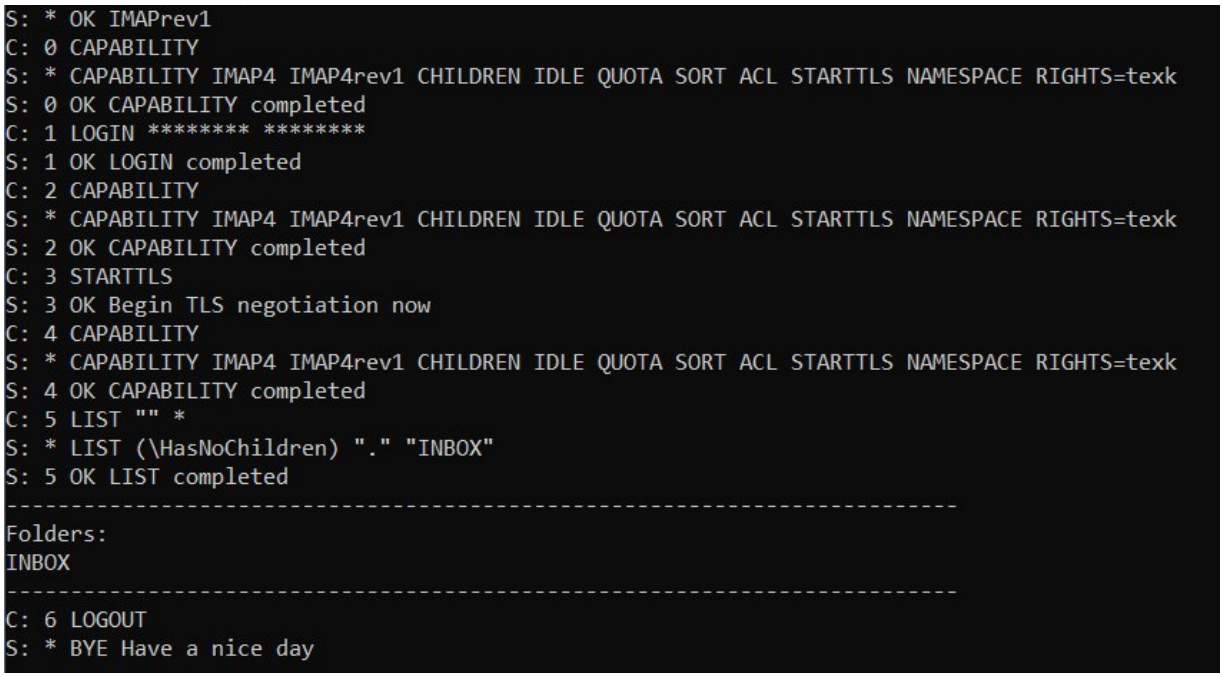

Рисунок 5.30 – демонстрація роботи команди StartTLS у консолі

5.4 Висновки до розділу

У даному розділі ми розглянули результати роботи проекту. Назвали вимогли для впровадження бібліотеки яка реалізує протокол IMAP4 в інші програми та системи та які переваги та функції вона дасть їм. Побачили як, описані у минулих розділах, діаграми та таблиці працюють в нашій бібліотеці. Також наглядно побачили як все таки працює протокол IMAP4 з реальною поштовою скринькою і який функціонал він має.

Висновки

В результаті проведеної роботи над моїм проектом було проведене ретельне вивчення роботи протоколу IMAP4. Він виявився дуже цікавим та функціональним. Цей протокол повністю відповідає сучасним реаліям та вимогам потенційних користувачів для роботи з електронною поштою.

Виконана мною робота має такі результати:

1) Проведений аналіз споріднених рішень реалізації протоколу IMAP4
2) Досліджена детальна документація по роботі протоколу IMAP4
3) Розроблена бібліотека мовою C++ для реалізації роботи стандартних команд протоколу IMAP4
4) Розроблені компоненти, які реалізують коректну роботу нашого протоколу на всіх рівнях з’єднання
5) Написано програмний код модуля системи Клієнта протоколу IMAP4
6) Розроблена консольна демонстрація можливостей бібліотеки
7) Випробувано програмну систему

Список використаних джерел

1. Герберт Шилдт. «C++: Базовый курс» 2010.

2. RFC (Request For Comment) 3501 Network Working Group. University of Washington 2003.

3. Майо, Джо. Самоучитель Microsoft Visual Studio 2010 : пер. с англ. / Дж. Майо. - Санкт-Петербург : БХВ-Петербург, 2011. - 464 с.

4. Рамбо Д. UML 2.0. Объектно-ориентированное моделирование и разработка / Д. Рамбо, М. Блаха. – Санкт-Петербург: Питер, 2007. – 544 с.

ДОДАТОК 1

Бібліотека методів C++ для роботи з сервером електронної пошти за протоколом IMAP 4

Специфікація

УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б

Аркушів 3

2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	Записка.docx	Пояснювальна записка
Комплекс		
Компоненти		
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPLIMAP4Client	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4MailBoxesListItem	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4MailBoxesList	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4MailBoxState	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4FetchResponceItem	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4FetchResponseItem	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4FetchResponse	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4MessageSet	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4AppendUIDResponse	Клас
УКР.НТУУ “КПІ	CPlIMAP4CopyUIDRespose	Клас

ім.І.Сікорського”. ТР62143_20Б		
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	CPlIMAP4Parser	Клас
УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б	Додаток 3	Опис програми

ДОДАТОК 2

Бібліотека методів C++ для роботи з сервером електронної пошти за протоколом IMAP 4

Текст програмного модуля

УКР.НТУУ “КПІ ім.І.Сікорського”.ТР62143_20Б

Аркушів 15

2020

```
        PL_IMAP4_CLIENT_H_INCLUDED
        PL_IMAP4_CLIENT_H_INCLUDED


         "PlStringList.h"
         "PlString.h"
         "PlDates.h"
         "PlTCPClient.h"
         "PlIMAP4Types.h"

        FACILITY_IMAP4  0xc4

        MAKEIMAP4ERROR(ErrCode) MAKEERROR(FACILITY_IMAP4, ErrCode)

const LONG IMAP4_E_FAILURE   = MAKEIMAP4ERROR(0x0001);
const LONG IMAP4_E_PROTOCOL   = MAKEIMAP4ERROR(0x0002);
const LONG IMAP4_E_BADREPLY   = MAKEIMAP4ERROR(0x0003);
const LONG IMAP4_E_TLSNOTSUPPORTED = MAKEIMAP4ERROR(0x0004);

//typedef CPlListTemplate<ElIMAP4NameListItem> CElIMAP4NameList_;

class CPlIMAP4Client;

// OnLog Event

typedef void (CPlEventTypesHolder::*PlIMAP4LogEvent)
 (CPlIMAP4Client* pSender, PlIMAP4Direction Direction, CPlString& strInfo);

class CElIMAP4LogEvent : public CPlMultiEvent
{
public:
 void Fire(CPlIMAP4Client* pSender, PlIMAP4Direction Direction,
  CPlString& strInfo)
 {
  for (int i = 0; i < m_Handlers.Count(); i++)
   ((m_Handlers[i]->m_Object)->*(PlIMAP4LogEvent)(m_Handlers[i]->m_Event))
    (pSender, Direction, strInfo);
 }
};

// OnLiteral Event

typedef void (CPlEventTypesHolder::*ElIMAP4LiteralEvent)
 (CPlIMAP4Client* pSender, CPlString &sToken, DWORD dwBytesCount,
  bool &bAsString, CPlStream **ppStream);

class CPlIMAP4LiteralEvent : public CPlEvent
{
```

```
public:
 void Fire(CPlIMAP4Client* pSender, CPlString &sToken, DWORD dwBytesCount,
  bool &bAsString, CPlStream **ppStream)
 {
  if ((m_Handler.m_Object != NULL) && (m_Handler.m_Event != NULL))
   ((m_Handler.m_Object->*(ElIMAP4LiteralEvent)m_Handler.m_Event))
    (pSender, sToken, dwBytesCount, bAsString, ppStream);
 }
};

// OnAlert Event

typedef void (CPlEventTypesHolder::*ElIMAP4AlertEvent)
 (CPlIMAP4Client* pSender, CPlString& strAlert);

class CElIMAP4AlertEvent : public CPlMultiEvent
{
public:
 void Fire(CPlIMAP4Client* pSender, CPlString& strAlert)
 {
  for (int i = 0; i < m_Handlers.Count(); i++)
   ((m_Handlers[i]->m_Object)->*(ElIMAP4AlertEvent)(m_Handlers[i]->m_Event))
    (pSender, strAlert);
 }
};

class CPlIMAP4MailBoxesListItem
{
public:
 DWORD dwAttributes; // IMAP4 name list item attributes
 CPlWideString strName;
 CPlWideString strDelimiter;
};

class CPlIMAP4MailBoxesList : public CPlList
{
protected:
 virtual int CompareItems(void *Item1, void *Item2);
public:
 CPlIMAP4MailBoxesList();
 virtual ~CPlIMAP4MailBoxesList();

 CPlIMAP4MailBoxesListItem *Get(int iIndex)
  { return (CPlIMAP4MailBoxesListItem*)CPlList::Get(iIndex); }
 int Add(CPlIMAP4MailBoxesListItem *pItem);
 virtual void Clear();
};

class CPlIMAP4MailBoxState
```

```
{
public:
 PlIMAP4MailBoxAccessMode eMode;
 DWORD dwTotalMessages;
 DWORD dwRecentMessages;
 DWORD dwUnseenMessages;
 DWORD dwFlags;
 DWORD dwPermanentFlags;
 DWORD dwNextUID;
 DWORD dwUIDValidity;
};

class CPlIMAP4FetchResponseItem
{
friend class CPlIMAP4FetchResponseLine;
private:
 PlIMAP4FetchResponseItemType m_Type;
 CPlString m_strName;
 CPlString m_strValue;
 bool m_bStreamed;
 DWORD m_dwStartPosition;
 DWORD m_dwLength;
public:
 CPlIMAP4FetchResponseItem(CPlString &strName, CPlString &strValue);
 void SetData(CPlString &strName, CPlString &strValue);

 PlIMAP4FetchResponseItemType GetType() { return m_Type; }
 void GetName(CPlString &strName) { strName = m_strName; }
 void GetRawValue(CPlString &strValue) { strValue = m_strValue; }
 void GetAsString(CPlString &strValue) { strValue = m_strValue; }
 long GetAsNumber();
 DWORD GetAsFlags();
 PlDateTime GetAsDateTime();
 bool IsStreamed() { return m_bStreamed; }
};

class CPlIMAP4FetchResponseLine : public CPlStringList
{
private:
 DWORD m_dwID;
public:
 CPlIMAP4FetchResponseLine(DWORD dwID);
 virtual ~CPlIMAP4FetchResponseLine();

 CPlIMAP4FetchResponseItem *Get(int iIndex)
  { return (CPlIMAP4FetchResponseItem*)CPlStringList::GetObject(iIndex); }
 int Add(CPlIMAP4FetchResponseItem *pItem)
  { return CPlStringList::AddObject(pItem->m_strName, pItem); }
 CPlIMAP4FetchResponseItem *GetByType(PlIMAP4FetchResponseItemType eType);
```

```
 CPlIMAP4FetchResponseItem *GetByName(CPlString strName);
 virtual void Clear();
 DWORD GetID() { return m_dwID; }
};


class CPlIMAP4FetchResponse : public CPlList
{
 friend class CPlIMAP4Client;
private:
 PlIMAP4IDType m_eIDType;
public:
 virtual ~CPlIMAP4FetchResponse();

 CPlIMAP4FetchResponseLine *Get(int iIndex)
  { return (CPlIMAP4FetchResponseLine*)CPlList::Get(iIndex); }
 int Add(CPlIMAP4FetchResponseLine *pItem)
  { return CPlList::Add(pItem); }
 virtual void Clear();

 PlIMAP4IDType GetIDType() { return m_eIDType; }
};

class CPlIMAP4MessagesSet : public CPlList
{
 friend class CPlIMAP4Client;
private:
 PlIMAP4IDType m_eIDType;

 virtual int CompareItems(void *Item1, void *Item2);
public:
 CPlIMAP4MessagesSet();
 virtual ~CPlIMAP4MessagesSet();

 DWORD Get(int iIndex);
 int IndexOf(DWORD dwValue);
 void Add(DWORD dwValue);
 void AddRange(DWORD dwFirst, DWORD dwLast);

 PlIMAP4IDType GetIDType() { return m_eIDType; }
 void SetIDType(PlIMAP4IDType eType) { m_eIDType = eType; }
};

class CPlIMAP4AppendUIDResponse
{
public:
 bool bValid;
 DWORD dwUIDValidity;
 CPlIMAP4MessagesSet cDstMessagesSet;
```

```
};

class CPlIMAP4CopyUIDResponse
{
public:
 bool bValid;
 DWORD dwUIDValidity;
 CPlIMAP4MessagesSet cSrcMessagesSet;
 CPlIMAP4MessagesSet cDstMessagesSet;
};

class CPlIMAP4Parser
{
private:
 CPlString m_sSource;
 CPlString m_sToken;
 bool m_bParenthesized;
 bool m_bSection;
 bool m_bHasMoreTokens;
 long m_lCurPos;
 long m_lBracketsCount;
protected:
 void SkipSpaces();
 void SkipUntilSpace();
 bool SkipUntilClosingQuote(char cQuote);
 bool SkipUntilClosingBracket(char cOpeningBracket, char cClosingBracket);
public:
 CPlIMAP4Parser();
 CPlIMAP4Parser(CPlString &sSource);
 ~CPlIMAP4Parser();

 void SetSource(CPlString &sSource);
 void GetSource(CPlString &sSource);

 bool HasMoreTokens();
 void GetCurrentToken(CPlString &sToken);
 void GetNextToken(CPlString &sToken);
 bool IsParenthesized() { return m_bParenthesized; }
 bool IsSection() { return m_bSection; }
 long GetCurrentPosition() { return m_lCurPos; }
 void GetRemainderString(CPlString &sString);
 void NextToken();
};

// IMAP4 Client Class

class CPlIMAP4Client : public CPlTCPClient
{
 friend class CPlIMAP4FetchResponseItem;
```

```
public:
 CPlIMAP4Client();
 virtual ~CPlIMAP4Client();

 virtual void Connect(CPlString& strHost, WORD wPort);

 void Capability(DWORD* pdwKnown, bool bSendRequest = false);

 void StartTLS();

 void Login();
 void Login(CPlString& strUserName, CPlString& strPassword);
 void Logout();
 void KeepAlive();

 void List(CPlWideString &strMailBox,
  CPlWideString &strRefName, CPlIMAP4MailBoxesList **ppMailBoxesList);
 void LSub(CPlWideString &strMailBox,
  CPlWideString &strRefName, CPlIMAP4MailBoxesList **ppMailBoxesList);

 void Select(CPlWideString& strMailbox, CPlIMAP4MailBoxState *pState);
 void Examine(CPlWideString& strMailbox, CPlIMAP4MailBoxState *pState);
 void Status(CPlWideString &strMailBox, DWORD dwDataItems,
  CPlIMAP4MailBoxState *pState);
 void Close();
 CPlIMAP4MailBoxState *GetLastMailBoxState() { return &m_MailBoxState; }

 void Fetch(CPlString &sDataItemNames, DWORD dwFirst, DWORD dwLast,
  bool bUID, CPlIMAP4FetchResponse **ppFetchResponse);
 void Search(bool bUID, CPlWideString& strCriteria, CPlString& strCharset,
  CPlIMAP4FetchResponse** ppFetchResponse);

 void Store(PlIMAP4StoreOperation eOperation, DWORD dwFlags,
  bool bSilent, DWORD dwFirst, DWORD dwLast, bool bUID,
  CPlIMAP4FetchResponse **ppFetchResponse);
 void Append(CPlWideString &strMailbox, DWORD dwFlags,
  PlDateTime &dtDateTime, bool bUseDateTime,
  CPlStream *pStream, DWORD dwBytesCount = 0,
  CPlIMAP4AppendUIDResponse **pAppendUIDResponse = NULL);
 void Copy(CPlWideString &destMailbox, DWORD dwFirst, DWORD dwLast,
  bool bUID, CPlIMAP4CopyUIDResponse **pCopyUIDResponse = NULL);
 void Expunge(CPlIMAP4FetchResponse **ppFetchResponse = NULL);
 void UidExpunge(DWORD dwFirst, DWORD dwLast,
  CPlIMAP4FetchResponse **ppFetchResponse = NULL);

 void Create(CPlWideString &strMailBox);
 void Rename(CPlWideString &strMailBox,
  CPlWideString &strNewName);
 void Delete(CPlWideString &strMailBox);
```

```
 void Check();
 void Subscribe(CPlWideString &strMailBox);
 void Unsubscribe(CPlWideSt
ring  &strMailBox);

 void GetLastError(CPlString& strError)
  { strError = m_strError; }

 DWORD GetLogDetails()
  { return m_dwLogDetails; }
 void SetLogDetails(DWORD dwValue)
  { m_dwLogDetails = dwValue; }

 PlIMAP4State GetClientState()
  { return m_State; }

 void GetPassword(CPlString& strValue)
  { strValue = m_strPassword; }
 void SetPassword(CPlString& strValue)
  { m_strPassword = strValue; }

 void GetUserName(CPlString& strValue)
  { strValue = m_strUserName; }
 void SetUserName(CPlString& strValue)
  { m_strUserName = strValue; }

 CPlWideString GetCurrentMailbox() {
  return m_strCurrentMailbox;
 }

 CElIMAP4LogEvent OnLog;
 CPlIMAP4LiteralEvent OnLiteral;
 CElIMAP4AlertEvent OnAlert;
protected:
 void DoLog(PlIMAP4Direction Direction, PlIMAP4LogDetail Detail, CPlString&
strInfo);

 HRESULT ExecuteQuery(const CPlString& strQuery, bool bLogOutput = true,
  bool bLogInput = true);
 HRESULT SendCommand(CPlString& strCommand, bool bLog = true);
 HRESULT ReceiveText(bool bLog = true);

 PlIMAP4Token ParseResponse();
 PlIMAP4Token ParseSection(CPlIMAP4Parser &parser);
 HRESULT ProcessLiteral(CPlString &str, bool &bLiteral, bool bLog);
 static bool IsLiteralToken(CPlString &str, DWORD *pdwBytesCount);
 void ParseNumber(CPlIMAP4Parser &parser, DWORD &dwNumber, DWORD dwDefValue);
 void ParseDateTime(CPlIMAP4Parser &parser, PlDateTime &dtDateTime);
```

```
 void ParseCapabilities(CPlIMAP4Parser &parser);
 void ParseListItem(CPlIMAP4Parser &parser);
 static void ParseFlags(CPlIMAP4Parser &parser, DWORD &dwFlags,
  bool bParenthesized = true);
 void ParseStatus(CPlIMAP4Parser &parser);
 void ParseFetch(DWORD dwID, CPlIMAP4Parser &parser);
 void ParseMessagesSet(CPlIMAP4Parser &parser,
  CPlIMAP4MessagesSet *pMessagesSet);
 void ParseAppendUID(CPlIMAP4Parser &parser);
 void ParseCopyUID(CPlIMAP4Parser &parser);
private:
 void ExamineSelect(DWORD dwCommand, CPlWideString& strMailbox,
  CPlIMAP4MailBoxState *pState);

 void MessageFlagsToString(DWORD dwFlags, CPlString &sValue);
 void MessageSetToString(DWORD dwFirst, DWORD dwLast, bool bUID,
  char *sCommand, CPlString &sValue, bool bAppendSpace);
 void MessageSetToString(CPlIMAP4MessagesSet *pMessagesSet,
  char *sCommand, CPlString &sValue, bool bAppendSpace);
private:
 DWORD m_dwLogDetails;
 //CElConverter* m_ToUnicode;
 //CElConverter* m_FromUnicode;
 PlIMAP4State m_State;
 PlIMAP4Token m_LastToken;
 DWORD m_dwCounter;
 CPlIMAP4Parser m_Parser;
 DWORD m_dwCapabilitiesCount;
 DWORD m_dwCapabilities;
 CPlStringList m_lstCapabilities;
 CPlIMAP4MailBoxesList m_ListResult;
 CPlWideString m_strCurrentMailbox;
 CPlIMAP4MailBoxState m_MailBoxState;
 CPlIMAP4MailBoxState *m_pMailBoxState;
 bool m_FetchResponseExpected;
 CPlIMAP4FetchResponse m_FetchResponse;
 CPlIMAP4AppendUIDResponse m_AppendUIDResponse;
 CPlIMAP4CopyUIDResponse m_CopyUIDResponse;

 CPlString m_strError;
 CPlString m_strLastTag;
 CPlString m_strPassword;
 CPlString m_strUserName;
};

BOOL IsIMAP4Error(HRESULT hResult);

void ConvertToUnicode(CPlString& Source, CPlWideString& Dest);
void ConvertFromUnicode(CPlWideString& Source, CPlString& Dest);
```

```
// IMAP4.cpp : This file contains the 'main' function. Program execution
begins and ends there.
//

         <iostream>
         <stdio.h>
         <stdlib.h>
         <io.h>
         <fcntl.h>
         "PlIMAP4Client.h"

CPlSocketEngine g_pSocketEngine;

class CIMAPEvents : CPlEventTypesHolder
{
public:
    void OnIMAP4Log(CPlIMAP4Client* pSender,
        PlIMAP4Direction eDirection, CPlString& sInfo)
    {
        if (eDirection == imapSent)
            wprintf(L"%S", "C: ");
        else
            wprintf(L"%S", "S: ");
        wprintf(L"%S", (char*)sInfo);
        wprintf(L"%S", "\n");
    };
};

CIMAPEvents g_Log;

int main()
{
    CPlIMAP4Client client;
    CPlIMAP4MailBoxesList *boxes = new CPlIMAP4MailBoxesList();
    CPlIMAP4MailBoxState mailBoxState;
    CPlIMAP4FetchResponse *fetchResponse = new CPlIMAP4FetchResponse();

    _setmode(_fileno(stdout), _O_U16TEXT);
    AddMultiEventHandler(&(client.OnLog), &g_Log, CIMAPEvents::OnIMAP4Log);

    CPlWideString mailBox(L"");
    CPlWideString refName(L"*");

    CPlString localUserName("olha@localhost");
    CPlString localPassword("1808");
    CPlString localHost("localhost");

    client.SetUserName(localUserName);
    client.SetPassword(localPassword);
```

```
    client.Connect(localHost, 143);
    client.Login();

    client.List(mailBox, refName, &boxes);

    wprintf(L"-------------------------------------------------------------
-----------\n");
    wprintf(L"Folders:\n");
    for (int i = 0; i < boxes->Count(); i++)
        wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
    wprintf(L"-------------------------------------------------------------
-----------\n");

    client.Logout();

    client.Connect(localHost, 143);
    client.Login();
    client.StartTLS();

    client.List(mailBox, refName, &boxes);

    wprintf(L"-------------------------------------------------------------
-----------\n");
    wprintf(L"Folders:\n");
    for (int i = 0; i < boxes->Count(); i++)
        wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
    wprintf(L"-------------------------------------------------------------
-----------\n");

    client.Logout();

    CPlString userName("                    ");
    CPlString password("eupacauitorbvnmr");
    CPlString host("              ");

    client.SetUserName(userName);
    client.SetPassword(password);
    client.SetSSLPassThrough(FALSE);
    client.Connect(host, 993);
    client.Login();

    client.List(mailBox, refName, &boxes);

    wprintf(L"-------------------------------------------------------------
-----------\n");
    wprintf(L"Folders:\n");
    for (int i = 0; i < boxes->Count(); i++)
        wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
```

```
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Examine(boxes->Get(2)->strName, &mailBoxState);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Selected folder (for read only) %s\n",
(wchar_t*)client.GetCurrentMailbox());
    wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
    wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    CPlString fetchItemNames("RFC822.SIZE BODY[HEADER.FIELDS (SUBJECT)]
INTERNALDATE FLAGS UID");

    client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    for (int i = 0; i < fetchResponse->Count(); i++)
    {
        CPlString messageSubject;
        fetchResponse->Get(i)->GetByName("BODY[HEADER.FIELDS (SUBJECT)]")-
>GetAsString(messageSubject);
        messageSubject.TrimRight();
        wprintf(L"Message %i size: %i subject: %S\n", i, fetchResponse-
>Get(i)->GetByName("RFC822.SIZE")->GetAsNumber(),
            (char*)messageSubject);
    }
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Select(boxes->Get(2)->strName, &mailBoxState)
;

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
    wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
    wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
```

```
    for (int i = 0; i < fetchResponse->Count(); i++)
    {
        CPlString messageFlags;
        fetchResponse->Get(i)->GetByName("FLAGS")->GetAsString(messageFlags);
        wprintf(L"Message %i size: %i flags: %S\n", i, fetchResponse->Get(i)-
>GetByName("RFC822.SIZE")->GetAsNumber(),
            (char*)messageFlags);
    }
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Store(imapRemove, imapSeenFlag, true, 0, imapLastMessage, false,
NULL);

    client.Status(boxes->Get(2)->strName, imapMessages | imapUnseen |
imapRecent, &mailBoxState);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
    wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
    wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
    wprintf(L"Unseen messages %i\n", mailBoxState.dwUnseenMessages);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Store(imapAdd, imapSeenFlag, true, 0, imapLastMessage, false,
NULL);

    client.Status(boxes->Get(2)->strName, imapMessages | imapUnseen |
imapRecent, &mailBoxState);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
    wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
    wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
    wprintf(L"Unseen messages %i\n", mailBoxState.dwUnseenMessages);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    CPlWideString mailBoxName(L"[Temporary]");
    CPlWideString mailBoxNewName(L"[Temporary New]");

    //client.Delete(mailBoxName);
    //client.Delete(mailBoxNewName);

    client.Create(mailBoxName);
```

```
    client.List(mailBox, refName, &boxes);
    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Folders:\n");
    for (int i = 0; i < boxes->Count(); i++)
        wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Copy(mailBoxName, 1, 2, false, NULL);

    client.Select(mailBoxName, &mailBoxState);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Selected folder %s\n", (wchar_t*)client.GetCurrentMailbox());
    wprintf(L"Total messages %i\n", mailBoxState.dwTotalMessages);
    wprintf(L"Recent messages %i\n", mailBoxState.dwRecentMessages);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    for (int i = 0; i < fetchResponse->Count(); i++)
    {
        CPlString messageSubject;
        fetchResponse->Get(i)->GetByName("BODY[HEADER.FIELDS (SUBJECT)]")-
>GetAsString(messageSubject);
        messageSubject.TrimRight();
        wprintf(L"Message %i size: %i subject: %S\n", i, fetchResponse-
>Get(i)->GetByName("RFC822.SIZE")->GetAsNumber(),
            (char*)messageSubject);
    }
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Rename(mailBoxName, mailBoxNewName);

    client.
List(mailBox, refName, &boxes);
    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Folders:\n");
    for (int i = 0; i < boxes->Count(); i++)
        wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
    wprintf(L"-----------------------------------------------------------------
-----------\n");
```

```
    client.Select(mailBoxNewName, &mailBoxState);

    PlDateTime dateTime = 0;
    CPlFileStream stream;
    stream.Open((TCHAR*)L"1.eml", false, true, false, true, true);
    client.Append(mailBoxNewName, 0, dateTime, false, &stream);
    stream.Close();

    client.Fetch(fetchItemNames, 0, imapLastMessage, false, &fetchResponse);

    wprintf(L"-----------------------------------------------------------------
-----------\n");
    for (int i = 0; i < fetchResponse->Count(); i++)
    {
        CPlString messageSubject;
        fetchResponse->Get(i)->GetByName("BODY[HEADER.FIELDS (SUBJECT)]")-
>GetAsString(messageSubject);
        messageSubject.TrimRight();
        wprintf(L"Message %i size: %i subject: %S\n", i, fetchResponse-
>Get(i)->GetByName("RFC822.SIZE")->GetAsNumber(),
            (char*)messageSubject);
    }
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Delete(mailBoxNewName);

    client.List(mailBox, refName, &boxes);
    wprintf(L"-----------------------------------------------------------------
-----------\n");
    wprintf(L"Folders:\n");
    for (int i = 0; i < boxes->Count(); i++)
        wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
    wprintf(L"-----------------------------------------------------------------
-----------\n");

    client.Logout();
}
```

ДОДАТОК 3

Бібліотека методів C++ для роботи з сервером електронної пошти за протоколом IMAP 4

Опис програмного модуля

УКР.НТУУ “КПІ ім.І.Сікорського”. ТР62143_20Б

Аркушів 6

2020

АНОТАЦІЯ

Дана програмна система призначена реалізації поштового протоколу IMAP4 у вигляді бібліотеки методів.

Створена за допомогою середовища розробки Microsoft Visual Studio 2019. Мова програмування – C++.

Існують такі переваги цієї системи: вона повністю реалізує протокол IMAP4 та всі його функції, встановлює безпечне з’єднання з сервером, зчитаю та коректно обробляє відповіді сервера в класи, коректно реалізовує команди клієнта протоколу, дозволяє працювати та виконувати різні дії з поштовою скринькою, поштовими папками та листами, оскільки ця система є лише частиною більшого продукту, то в повній системі вона має різні сценарії для використання, у наш проект також входить демонстрація можливостей у вигляді консолі, яка працює з реальними поштовими серверами.

ЗМІСТ

-4-

1 ЗАГАЛЬНІ ВІДОМОСТІ

Назва програмного модуля демонстрації – ““IMAP4.cpp”. Мова програмної реалізації – C++. Середовище розробки – Microsoft Visual Studio 2019.

Використовується для демонстрації можливостей бібліотеки методів протоколу IMAP4.

Для її використання необхідно:

- комп’ютер або ноутбук;
- операційна система Windows;
- встановлений середовище Microsoft Visual Studio 2019;

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Бібліотека розроблена як частина великого проекту, вона лише реалізує стандартний протокол IMAP4 з усіма його вимогами та стандартами.

Та при цьому сценаріїв використання цього модуля його безліч, так як і стандартних команд клієнта. Команди реалізовують всі можливості даного протоколу, а кожна команда це по суті функція, яка може стати новою додатковою чи основною функцією іншої програми.

-5-

3 ВХІДНІ ДАНІ

Вхідними даними нашої програми можна вважати команди які клієнт надсилає серверу, у нашій демонстрації ми прописуємо ці команди та їхні аргументи у коді.

Ось наприклад на рисунку 3.1 команда Rename та List.

```
client.Rename(mailBoxName, mailBoxNewName);

client.List(mailBox, refName, &boxes);
wprintf(L"--------------------------------------------------------------------------------\n");
wprintf(L"Folders:\n");
for (int i = 0; i < boxes->Count(); i++)
    wprintf(L"%s\n", (wchar_t*)boxes->Get(i)->strName);
wprintf(L"--------------------------------------------------------------------------------\n");
```

Рисунок 3.1 – Приклад вхідних даних у вигляді коду

Потім в консолі ці рядки коду перетворюються в стандартний запис команди клієнта, як на рисунку 3.2.

```
--------------------------------------------------------------------------------
C: 16 RENAME "[Temporary]" "[Temporary New]"
```

Рисунок 3.2 – Приклад вхідних даних у консолі

-6-

4 ВИХІДНІ ДАНІ

Вихідними даними є відповіді сервера на надіслані клієнтом команди. Вони стандартизовані але надсилаються хаотично і наша бібліотека вміє їх зчитувати та правильно розуміти за допомогою парсера, записувати їх у класи і потім виводити у вигляді стандартизованих відповідей на команди клієнта як у звичному нам записі протоколу.

Наприклад відповідь до наших вхідних даних команд Rename та List на рисунку 4.1.

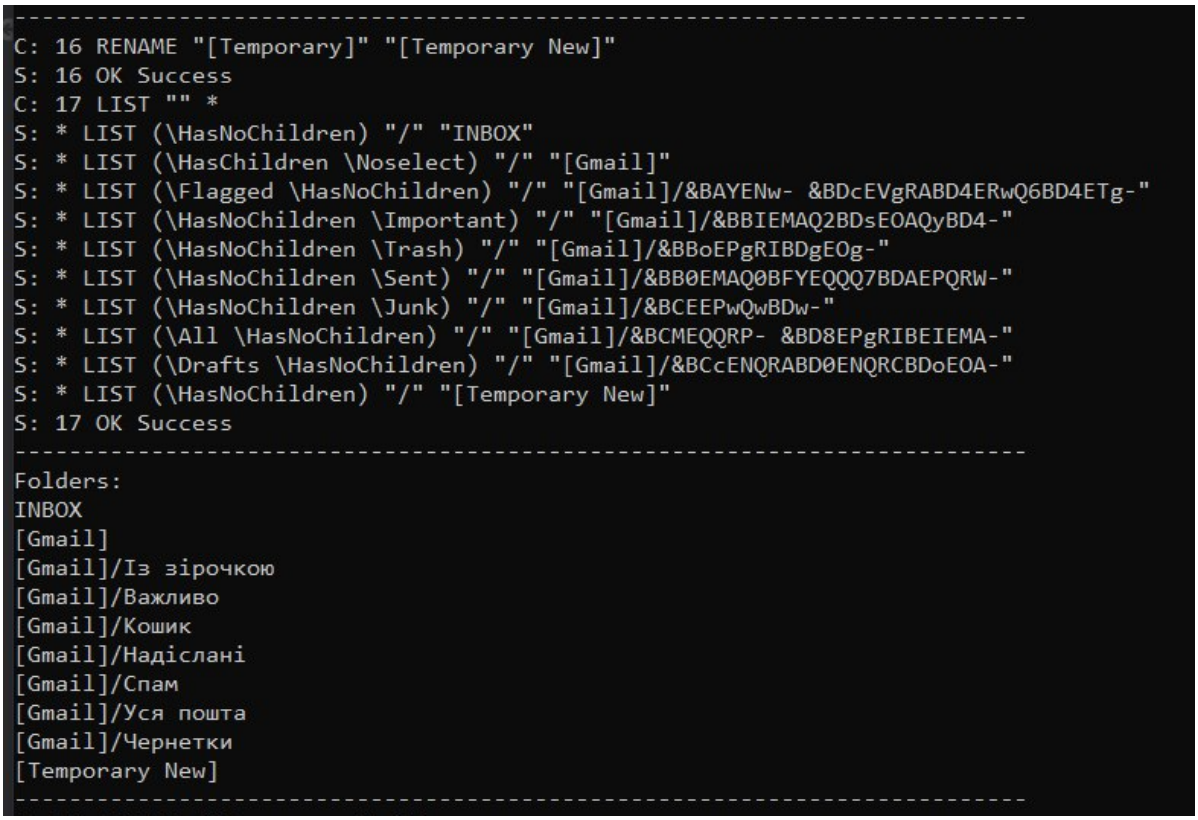

Рисунок 4.1 – Приклад вихідних даних